The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! May 1986: Volume 6, Issue 06. $3.50

Real Life Applications

- The Amiga: A Programmer's Perspective,
- The Amiga: A User's Perspective
- Amiga Reference Tables
- Crystal Ball: Predict Team Sports Outcomes And Track Player Performance
- Drop-Down Menus Utility For The C64
- Comparison of 4 Wordprocessors
- DVORAK Keyboard For The 64
- Screenboard: A Joystick Controlled Keyboard Utility
- Hardware Project: VIC 20 Total Home Controller

0 72246 00594 05

THE TIME SAVER

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only $7.95 Per Issue
6 Disk Subscription (one year)
Just $45.00
(see order form at center fold)

Volume 6
Issue 06
Circulation 64,000

The Transactor

Real Life Applications

Start Address Editorial 3

Bits and Pieces 5
SAVERIFY
Double Verifizer
Corrupting RAMTAS Update
Finding the Missing File
LOAD & RUN Trick
Check For Device Present
Word-Wrap For VIC, 64, PET, etc.
Visible "searching" Messages
C-64 Scroll Down Routine
Easy 'RESTORE x'
Sneaky Saves
Sanitation Engineer: Faster Garbage Collection?
C128 Bits
Ornament and Happy New Year In High-Res
Multiple Circle, Triangle and Square High Res Draw Routine
Incredible 3-D Effect High Res Draw Routine
More Ideas
Amiga Bits: Some Notes About CLI

Letters 11
1200 BPS Response
More Responding at 1200 BPS
Almost Clear
Transactor/Ohio Porting
A BIT Of A Problem
Left Wing Interference
The Gremlin Effect
Jordan Rolltop Stand Revisited
The Horror Of Hex
False ID
Attack Of The Killer Clone

News BRK 75
The TransBASIC Disk
PAL and POWER: The ToolBox
The G-Link Interface
Attention Anthology Owners
World Of Commodore III In Toronto A Success
C.A.S.E Meeting And Jamboree 1986
NAAUG For Amiga Users
BBS For Amiga Owners
dBx Translator Converts dBASE Programs Into "C".
The Trading Board BBS
2890 Databases Available Online
Tymnet Offers Local Access Service In Canada
Basic Compiler For The C128
Chartpak 128 for the Commodore C128
Statistical Programs for Commodores
Bookkeeper's Aid
Wilanta Descender ROM
Attention B Machine Owners: 1 Meg RAM Board

TransBASIC Installment #8 Modules So Far on page 22 19
The Amiga: A User's Perspective 29
The Amiga: A Programmer's Perspective 33
Amiga DOS and CLI Commands A handy quick reference . . 38
Amiga Editor Commands Using "ED", the screen oriented editor . . 42
EDIT: Amiga's Line Oriented Editor 43
Pick Areas and Pop Menus Drop-down menus for the C64 44
DVORAK 64 Convert your keyboard through software 49
Screenboard Complete keyboard control from a joystick 52
Crystal Ball Make team sports predictions with this statistics analyzer 56
Home Controller Run your entire house from a VIC 20 62
A Comparison of 4 Wordprocessors 72
Compu-toons 79

Note: Before entering programs, see "Verifizer" on page 4

The Transactor
The Tech/News Journal For Commodore Computers

Editor in Chief
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Lana Humphries

Contributing Writers

Ian Adam
Daniel Bingamon
Anthony Bryant
Tim Buist
Jim Butterfield
Gary Cobb
Jeffery Coons
Pierre Corriveau
Bob Davis
Elizabeth Deal
Yijun Ding
Michael J. Erskine
Jim Grubbs
Tom Hall
Bob Hayes
John Jay Hilfiger
Andy Hochheimer
John Holttum
Chris Johnson
Mark Jordan
Gary Kiziak
Jesse Knight
James E. LaPorte
William Levak
Jack Lothian
Scott Maclean
Jim McLaughlin
Terry Montgomery
Michael Mossman
Gerald Neufeld
Noel Nyman
Dave Pollack
Richard Perrit
Terry Pridham
Raymond Quirling
Glen Reesor
Gary Royal
John W. Ross
Louis F. Sander
Fred Simon
Perry Shultz
Edward Smeda
Darren J. Spruyt
Nick Sullivan
Zoltan Szepesi
Karel Vander Lugt
Audrys Vilkas
Andrew Walduck
Steven Walley
Jack Weaver
Charles Whittern
Chris Wong

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print " flush right " – would be shown as – print "[10 spaces]flush right "

Cursor Characters For PET / CBM / VIC / 64

Down – [q]
Up – [Q]
Right – [ ]
Left – [Lft]
RVS – [r]
RVS Off – [R]
Insert – [ ]
Delete – [t]
Clear Scrn – [ ]
Home – [s]
STOP – [c]

Colour Characters For VIC / 64

Black – [P]
White – [e]
Red – [ ]
Cyan – [Cyn]
Purple – [Pur]
Green – [ ]
Blue – [ ]
Yellow – [Yel]
Orange – [A]
Brown – [ ]
Lt. Red – [ ]
Grey 1 – [ ]
Grey 2 – [ ]
Lt. Green – [ ]
Lt. Blue – [ ]
Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – [E]
F2 – [ ]
F3 – [F]
F4 – [J]
F5 – [G]
F6 – [K]
F7 – [ ]
F8 – [ ]

Please Note: The Transactor has a new phone number: (416) 878 8438

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

Quantity Orders:

U.S.A. Distributor:
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 941 7911

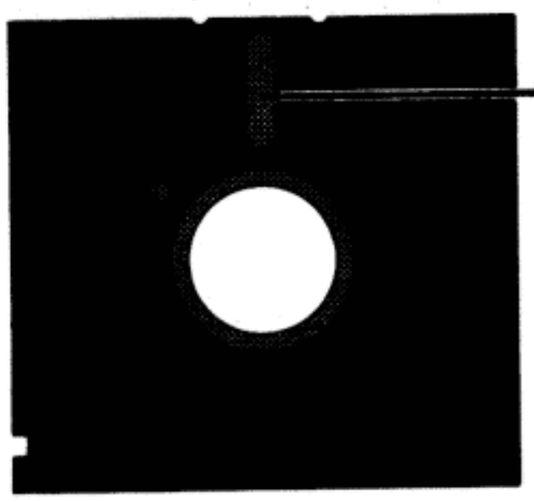
Norland Communications
251 Nipissing Road, Unit 3
Milton, Ontario
L9T 4Z5
416 876 4774

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 04, 05, 06, Vol 5 Issues 03, 04
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05

Back Issues: $4.50 each. Order all back issues from Milton HQ.

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package.
The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Start Address

SoftBound

What is happening?! Creative Computing, 11 years publishing, gone. Popular Computing, gone. InfoAge, Commander, Micro, Kilobaud, Software Now, all gone. This list goes on, too, into areas outside those covering Commodore equipment. One report I recall stated "out of 160 computer magazines that were publishing in 1982, only 35 remain today". The common complaint is a lack of advertising revenue due to a "soft" market.

In his closing editorial, *Creative's* David Ahl complained that most advertising had become a decision based on "lowest cost per thousand readers". But I think the industry is getting smarter than that. *InfoAge's* Gordon Campbell does too. "InfoAge suffered from not being able to deliver a stable market to our advertisers", we became too "heterogenous" as quoted from *Toronto Computes*. Vertical magazines offer the reader more reading per dollar. So it's only natural that more readers, who are also getting smarter, will be attracted to the higher concentration. Indications are that many firms are spending their ad dollars in magazines that have a tighter focus on the industry. Ahl's complaint becomes one of semantics. If only a fraction of each thousand readers can be qualified, cost per thousand goes way up.

However, I admit the horizontal publications couldn't just pull a brand from a hat, and focus. They could have, but they waited too long. I also suspect not enough deflection on the 'lean and trim' meter. Because what confuses me most is that all the above mags *did* have ads. Almost half of Creative's 96 page final was advertising!

Now I don't claim we are immune to the same fate because we seem to be afloat without ad revenue. Not at all. For the last 9 issues we have depended on nothing but sales. Distributors have told us, "publishers would sell their souls for your stick rate at the news stand". But it just doesn't seem to be enough. Is this an S.O.S.? No. But if you know a place where one more Transactor might be sold, you can help. Write their name and address on our postage paid subscription form and check off the box marked "please send dealer information". You can bet we'll get in touch with them, but please use discretion. In the magazine distribution business you soon learn that not every smoke shop and convenience store is a suitable venue. We're looking for outlets that specialize in reading material as much as we do. And who knows – you may just help yourself get Transactor a little closer to home.

Why all the malarky about advertising? We have other plans too. The Transactor has decided to offer ad space. Several requests have been denied over the last 9 issues, as will several more over those to come. Since we *do* seem to be surviving without ad revenue, we're going to be very selective about the ads we accept. Our standards are high and we have a keen sense on our focus. Advertising of little or no value to our readers will not be considered. In fact, we would have to feel good enough about the ad to want to sell that product or service. Which leads me to the next phase of our plan.

With prices dropping everywhere except in ad rates, it comes as no surprise that advertisers are indeed "tightening up". Retail prices can only drop so low. The manufacturer, distributor, and retailer all take their percentage and when it stops adding up, they all feel the pinch. On the other hand, the mail-order business has very low operating costs which is ideally suited to items priced low enough to move. You may have noticed we're selling The Toolbox through mail-order at $50 off suggested retail. We like this package and if you think we'll like yours as much, talk to us. Your percentage wouldn't change, but the possibility of subsidizing the cost of advertising with a merchandise exchange is open for discussion. You could look me up in Toronto directory assistance, but I'll save you the trouble. It's 416 221 2922 any time.

Does this mean the end of the 95% Ad Free Transactor? No way. Only 7 pages will be available to start – the cover spots and a four page glossy insert at the center of the magazine. 7 over 88 makes only 8% advertising, and if response warrants more ad space, the editorial content will be increased to maintain the ratio. Colour ads will get higher preference, but again, only if they meet with criteria.

By expanding our horizons The Transactor will become even more insulated from the ad space insertion order. Magazine quality control will propagate through to our mail-order operation where we have the opportunity to strengthen our sales, help stimulate the industry we believe in, and at the same time offer a trustworthy service to our readers.

In brief, our 20/20 deal still stands – order 20 anything for 20% off. Our Viewtron section should be operational by February – sign on and enter "transactor", and we'll have a complete explanation for you next issue. The "ultra-glazed" cover means you can casually wipe off any coffee spills, and you're looking at our very first experiment with "spot colour". And yes, it looks like there will be another Inner Space Anthology, but not for at least another 5 months. We're making this early notice mainly for your input. If there's a table, listing, chart, diagram, index, illustration, or any other skiagraphical cartographic images you may have, let us know. Often they can be computer generated and my typesetting instincts are down to a reflex. Anyone who is aware of errors in the first edition is invited to send them along. We have a couple copies of published magazine reviews (TPUG, ICPUG, PCA) but if you know of others, a copy would be sincerely and personally appreciated.

Some of the items we have planned for the next Anthology include new material for the 68000 and the Amiga, C128 stuff, Z80/Z80B specs and CP/M for posterity, 1571 notes, MS DOS commands, a modem section, more printer info, more hardware specs, an updated BBS and club listing, DataPac and other network parameter settings plus a listing of time-sharing call addresses. Tentatively planned are Jim Butterfield's commented disassembly of the C64 and C128. They're pretty long so I'll be forced to reduce the type to near microscopic, but even then they may consume too much space. I'm also considering a table of popular guitar chords and their finger positioning – as my choice for the "wonder-why-that's-there" category.

There *is* nothing as constant as change, I remain

Karl J.H. Hildon, Editor In Chief

P.S. Could someone help me? I'm looking for engineering software for any CBM ie. calculating forces, analytical geometry, etc.

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two–letter code for each program line which you can check against the corresponding code in the program listing.

There are two versions of VERIFIZER on this page; one is for the PET, the other for the VIC or 64. Enter the applicable program and RUN it. If you get the message, "***** data error *****", re–check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 828 to enable the C64/VIC version (turn it off with SYS 831)
or SYS 634 to enable the PET version (turn it off with SYS 637)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing it means we've editted that line at the last minute which changes the report code. However, this will only happen occasionally and only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re–check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors (eg. POKE 52381,0 instead of POKE 53281,0), but ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm–start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

Listing 1a: VERIFIZER for C64 and VIC–20

```
KE  10 rem* data loader for " verifizer " *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print " ***** data error ***** ": end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP  1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC  1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN  1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG  1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM  1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA  1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG  1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK  1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN  1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH  1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC  1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP  1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH  1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH  1140 data 101,  89, 133,  89,  96
```

Listing 1b: PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI  10 rem* data loader for " verifizer 4.0 " *
CF  15 rem pet version
LI  20 cs=0
HC  30 for i=634 to 754:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
OG  60 if cs<>15580 then print " ***** data error ***** ": end
JO  70 rem sys 634
AF  80 end
IN  100 :
ON  1000 data  76, 138,   2, 120, 173, 163,   2, 133, 144
IB  1010 data 173, 164,   2, 133, 145,  88,  96, 120, 165
CK  1020 data 145, 201,   2, 240,  16, 141, 164,   2, 165
EB  1030 data 144, 141, 163,   2, 169, 165, 133, 144, 169
HE  1040 data   2, 133, 145,  88,  96,  85, 228, 165, 217
OI  1050 data 201,  13, 208,  62, 165, 167, 208,  58, 173
JB  1060 data 254,   1, 133, 251, 162,   0, 134, 253, 189
PA  1070 data   0,   2, 168, 201,  32, 240,  15, 230, 253
HE  1080 data 165, 253,  41,   3, 133, 254,  32, 236,   2
EL  1090 data 198, 254,  16, 249, 232, 152, 208, 229, 165
LA  1100 data 251,  41,  15,  24, 105, 193, 141,   0, 128
KI  1110 data 165, 251,  74,  74,  74,  74,  24, 105, 193
EB  1120 data 141,   1, 128, 108, 163,   2, 152,  24, 101
DM  1130 data 251, 133, 251,  96
```

Bits and Pieces

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits & Pieces column, we'll credit you in the column and send you a free one–year's subscription to The Transactor

SAVERIFY

Bob Hayes
Winnipeg, Manitoba

This is a short program which enables the 64 (and possibly other Commodore?) owner to SAVE and VERIFY a program with one command. The format is:

SYS(address) " filename " ,8

Where the address is the start of the machine language code (relocatable in the BASIC loader).

Here is the assembly:

```
start   jsr  $e1d4
        jsr  $e159
        lda  #$01
        sta  $93
        bit  $00a9
        sta  $0a
        jsr  $e16f
        rts
```

And the BASIC loader:

```
10 rem ** saverify -- bob hayes **
20 rem ** wpg, man. canada      **
25 sa = 828: rem start address–note: relocatable
30 q$ = chr$(34): a = 0: for x = 0 to 18: read q: poke sa + x,q
    : a = a + q: next
40 print: print " format: sys " sa;q$ " filename " q$ " ,8 "
50 end
60 data 32,212,225,32,89,225,169,1,133,147,
        44,169,0,133,10,32,111,225,96
```

Double Verifizer

Steven Walley
Sunnymead, CA

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifizer–displayed codes. The program below, 'DOUBLE VERIFIZER' solves that problem by showing the two–letter verifizer code on both the first and second row of the TV screen. The program uses the interrupt vector to update the screen every 1/60 of a second.

To use Double Verifizer, just run the below program once the regular Verifizer is activated.

```
KM  100 for ad = 679 to 720:read da:poke ad,da:next ad
BC  110 sys 679: print: print
DI  120 print " double verifizer activated " :new
GD  130 data 120, 169, 180, 141,  20,   3
IN  140 data 169,   2, 141,  21,   3,  88
EN  150 data 96,  162,   0, 189,   0, 216
KG  160 data 157,  40, 216, 232, 224,   2
KO  170 data 208, 245, 162,   0, 189,   0
FM  180 data   4, 157,  40,   4, 232, 224
LP  190 data   2, 208, 245,  76,  49, 234
```

Corrupting RAMTAS Update

Yijun Ding
Pittsburgh, PA

"Corrupting RAMTAS Routine" in Bits and Pieces Volume 6, issue 4 mentioned the fact that $A000 will contain $55 after a reset. But there is more. RAM from $FD30–$FD4F will be written with the contents of the corresponding ROM, as the routine at $FD15 ($FF8A, reset vectors in $0314–$0333) is called in a reset process. Actually, the RAM at $FD30–$FD4F will be corrupted every time $FD15 is called.

Finding the Missing File

Jeffery Coons
Lake Ridge, Virginia

If a program bombs because it needed some file that wasn't on the disk, you can find out what file the program wanted with this one–liner:

```
for i = 0 to peek(183)–1: poke 1024 + i,
peek(peek(188)*256 + peek(187) + i): next
```

The name of the last file used will be displayed at the top left-hand corner of the screen. You have to POKE to the screen in this manner because PRINTing will corrupt the last character in the string. Users with ROM version 2 will have to also POKE to colour memory (at 55296+i) or make sure there is some text already on the top line of the screen.

LOAD & RUN Trick

Chris Wong

. . .A really neat load and run trick: After you type

```
load " filename " ,8,1
or load " filename " ,8:
```

Press shifted RUN/STOP instead of RETURN. The program will automatically RUN itself after loading. It eliminates the old load/return/run/return routine, easing up loading a bit.

Check For Device Present

Dave Pollack
Commodore E. Brunswick Users Group (CEBUG), E. Brunswick ,NJ

As most every C-64 user knows, the 'DEVICE NOT PRESENT' message and consequent crash is not the most pleasant experience in the world to endure. Believe me, I've been searching for close to a year for ANY solution that will work. It was not that obvious. I stumbled upon it quite by accident after coding a small routine that provided a way for me to print the value of the 'ST' variable after multiple I/O operations. If you do that you'll notice something interesting. An OPEN followed by an immediate CLOSE will not hang the computer even if the device is not present, but it allows you to interrogate ST which returns a nonzero result in this case.

If you use the following code, your program will be able to check for DEVICE NOT PRESENT and continue without bombing.

```
100 open 15,8,15: close 15
110 if st<>-128 then 160
120   print " !! DRIVE NOT PRESENT !! "
130   print " ## check drive power and cables, then
            press a key ## "
140   get a$:if a$ = " " then 140: rem wait for a key
150   goto 100
160 rem program continues. . .
```

Word-Wrap For VIC, 64, PET, etc.

Gary Royal
Chicago, IL

There never seems to be enough columns on the screen to display what you want to print on it. And there's nothing uglier than a word hanging partly on the end of one line and at the beginning of another. Whining about it does no good (I've tried), but word-wrap does. Place the string you want wrapped in 'w$', the desired line width in 'w', and call this routine.

```
100 rem* recursive word-wrap routine *
110 rem* put string in w$,
120 rem* line width in w
130 :
140 if len(w$)>w then 160
150 print w$: return
160 p=0: for i=w to 1 step-1
170 if p=0 and mid$(w$,i,1)=" " then p=i
180 next: h$=right$(w$,len(w$)-p)
190 w$=left$(w$,p): gosub150
200 w$=h$: goto 140
```

Since strings in Microsoft BASIC can be up to 255 characters long, you can easily squeeze five screen lines into w$ with the peace of mind that can only come from the knowledge that it will be formatted legibly. But beware! the routine is recursive, and assumes that words in the text will be separated by spaces. If the length of w$ is greater than 'w' and 'w$' contains no spaces it will loop forever, so avoid hyphenated words that might be longer than your desired line length (or modify line 170 to look for hyphens, too).

Visible "searching" Messages

Terry Montgomery
Auckland, New Zealand

In direct mode you get 'SEARCHING' and 'FOUND' messages that tell you what is going into the computer. These messages can be extremely helpful, especially when using tape. But when LOAD statements are encountered in program mode, the messages are suppressed. During program development, it would be nice to see what's going on a bit more. Here are two ways to see these messages from a running program:

1) Use GOTO instead of RUN to start the program. If the first line is 0, GOTO doesn't need a line number specified.

2) POKE 157,128 to flag direct mode. This can be turned off by POKE 157,0. This way you can get messages from one part of the program and block them from others.

C-64 Scroll Down Routine

Chris Johnson
Toronto, Ont.

In Volume 5, Issue 2 of The Transactor, Paul Blair reported a ROM routine that scrolled down the screen of a C-64. He also mentioned that it "left some pointers a bit untidy . . . a PRINT or two seems to restore order".

I found that a PRINT or two did *not* set things right; however, resetting the screen line link table did. The following routine

clears the link table before and after calling the scroll–down routine.

The syntax to use is:

SYS address, n, topline

Where n is the number of times you want the screen to be scrolled down one line and topline (0 to 24) is the last line not to be scrolled. All the lines below this will be scrolled down x times.

To change the location of the routine, just change the value of s in line 110. The loader will make the necessary changes to the machine code.

```
AF  100 rem* c-64 scroll down *
MO  110 s=49152: rem start address (relocatable)
OL  120 for i=s to s+33: read a: poke i,a: next
PH  130 print" ** scroll down - syntax: "
DI  140 print" sys "s" ,n,topline"
LA  150 print" Where 'n' = number of lines to scroll"
EB  160 :
OI  170 if s=49152 then end
GM  180 u=s+22: ju=s+7: r=s+34: jr=s+4
CB  190 jj=s+18
AG  200 poke ju+1,u/256: poke ju,u-256*peek(ju+1)
PN  210 poke jj+1,r/256: poke jj,r-256*peek(jj+1)
BE  220 poke jr+1,r/256: poke jr,r-256*peek(jr+1)
KF  230 :
HB  240 data  32, 241, 183, 142,  34, 192,  32,  22
KD  250 data 192,  32, 241, 183, 134, 214,  32, 101
FH  260 data 233, 206,  34, 192, 208, 248, 162,  24
AD  270 data 181, 217,   9, 128, 149, 217, 202, 208
GJ  280 data 247,  96
```

Easy 'RESTORE x'
Using TransBASIC

Andy Hochheimer
Wallaceburg, Ont

I have been using a lot of DATA statements in programming for quite a while. 99% of the time I have to RESTORE then search for my data on a specific line number before reading again. In Transactor Volume 5 Issue 3 was this 'RESTORE x' program from Gary Kiziak, which allowed a RESTORE to a specific line number:

```
10 restr=828: for k=restr to restr+31:read j:poke k,j: next
20 data  32, 253, 174,  32, 158, 173,  32, 247
30 data 183,  32,  19, 166, 175,   5, 162,  17
40 data  76,  55, 164, 165,  95, 233,   1, 133
50 data  65, 165,  96, 233,   0, 133,  66,  96
60 rem format: sys restr x
```

I've found a shorter and easier way to RESTORE X, using TransBASIC:

10 doke 65,line(x)+4

This incredible program line *does* work; location 65 is the Current DATA Address. It restores the pointer to the first byte of line X. The 4 is added to avoid reading the last data element of the previous line. This is a small sample of the great things you can do with TransBASIC!

Sneaky Saves **Terry Pridham, Belmont, Ont.**

In Vol 5 issue 3, "Unveiling The Pirate Part 2: Programming Sleight of Hand" – 'Ye Olde Standbye', where by using a shifted–space before the filename within quotes produces a directory that shows two quotes followed by the filename:

save" 0:[Shift-space]filename" ,8

In the directory it becomes:

3 " "filename prg

By experimenting with it, I found even more ways to twist the minds of Pirates (as if they weren't in the first place). Ever see directories where the name of the program is in reverse field? Well here's how it's done. Type:

save,1 quote, drive number, colon, 1 quote, rvs on, 1 delete, 2 inserts, shift–M, rvs on, rvs off, filename, quote, comma, device number

When done, it will appear something like:

save " 0: Mr filename" ,8

In the directory, it will show the block count and the first quote where it would normally appear. The shifted M causes a carriage return (because a shifted reverse M is a '13') and the filename will appear right under the block count in reverse field. The file type indicator (i.e. "prg") and the spaces preceding it will also appear in reverse field.

Try adding a couple DELs, or even cursor control characters, by hitting 1 Insert for every control character you wish to include immediately before the filename. However, you must remember what characters are in this "prefix" in order to LOAD that file. Experiment and have fun!

Sanitation Engineer

Fred Simon
Gibbsboro, New Jersey

Did you ever have to wait for several minutes while your computer collected "garbage" strings? Garbage collection on the C-64 has been known to take more than twenty minutes when a large number of strings need be processed. With the program "Sanitation Engineer", active strings are collected lightning fast.

What Is Garbage Collection?

Each time the Basic interpreter encounters a new string variable definition, it builds that string character by character in high memory, working downward from location 40960. If a string variable is changed, the old string remains in memory as "garbage". If the available free memory is less than the maximum length of a couple of strings, or if the Basic command FRE(0) is issued, the garbage collection routine is called. This routine looks at each string variable to find the one stored highest in memory, moves all of the other strings down by the length of this string, and then copies the string to the top of available memory. The length of time it takes to complete this task depends only on the number of strings and not their length.

To see garbage collection at work, try this program:

```
10 d=500: dim x$(d)
20 for j=0 to d: x$(j)=str$(j): next
30 print "starting collection. . ."
40 t=ti: j=fre(0)
50 print (ti-t)/60" seconds"
```

Change the value of D in line 10 to see the effect of increasing the number of strings.

Faster Collection

One way to speed up garbage collection is to first copy the string memory to a buffer area (Sanitation Engineer uses the area located underneath the Kernal ROM). Each active string can then be pulled out of the buffer and written to the clean string area. The bottom of the string memory is then the bottom of the last active string copied from the buffer. Sanitation Engineer is written as a "patch" to the Basic operating system. It uses the area of memory from 51740-52223 for the garbage collection routines. Thus, it can be used with the DOS Wedge and leaves 49152-51739 free for other machine language routines.

Type in and Save Sanitation Engineer. A mistake in one of the Data statements could cause your computer to lock-up when the routine is executed. A checksum is included to reduce the chance of errors. When you Run the program, Basic ROM is first copied to RAM. The new address for the Sanitation Engineer is written over the old collection routine. In addition, the READY. prompt is changed to READY! to remind you that Basic has been modified. If you hit Run/Stop-Restore, the Sanitation Engineer will be deactivated. To reactivate, just type SYS 51740. Try the test program you typed in earlier. Change D to 5000 and try again. No more delays!!

Sanitation Engineer Basic Loader

```
PI  10 rem save"0:sanitation 64",8
DD  100 rem sanitation engineer
FN  110 rem for the commodore 64
BP  120 rem by fred simon  8/85
HO  130 ck=0: for i=51740 to 52223: read d
BP  140 poke i,d: ck=ck+d: next
EG  150 if ck=63591 then sys51740: end
GK  160 print "error in data statements": stop
OB  170 :
MD  180 data  120, 169,  55, 133,   1, 169, 160, 133
HL  190 data    3, 160,   0, 132,   2, 177,   2, 145
AN  200 data    2, 136, 208, 249, 230,   3, 165,   3
JH  210 data  201, 192, 208, 241, 169,  54, 133,   1
NC  220 data   88, 169,   5, 141, 143, 183, 169,  33
LG  230 data  141, 125, 163, 162,   2, 189,  83, 202
BF  240 data  157,  38, 181, 202,  16, 247,  96,  76
JK  250 data   86, 202, 169,   0, 141, 239, 203, 169
EG  260 data   15, 133, 250, 169, 224, 133, 249, 165
NI  270 data   52, 141, 240, 203,  56, 229,  50, 201
NE  280 data   19, 144,  22, 233,   3, 133, 250, 165
ND  290 data   50, 105,   0, 133, 249, 165,  56, 229
DB  300 data   52, 105,   1, 197, 250, 176,   2, 133
EK  310 data  250, 165,  56, 141, 242, 203, 165,  55
GD  320 data  141, 241, 203, 133,  51,  24, 240,   1
GI  330 data   56, 173, 242, 203, 133,  52, 233,   0
KJ  340 data  133, 251, 105,   0, 133, 252, 165,  50
KB  350 data  105,   1, 133, 254, 165,  45, 233,   6
HJ  360 data  133,  95, 165,  46, 233,   0, 133,  96
NK  370 data  165,  47, 133, 253, 165, 251, 205, 240
BL  380 data  203, 144,  51, 229, 250, 133, 248, 165
EA  390 data   52, 229, 251, 229, 248,  73, 255, 105
OK  400 data    2, 197, 248, 144,   2, 165, 248, 205
JI  410 data  240, 203, 176,   5, 173, 240, 203, 233
DI  420 data    0, 133, 251,  32, 138, 203, 166,  48
BO  430 data   32, 243, 202, 176,   9,  32,  39, 203
CC  440 data  165, 251, 133, 252, 144, 182,  96,  24
GC  450 data  165,  95, 105,   7, 133,  95, 144,   2
KF  460 data  230,  96,  69,  47, 208,   4, 228,  96
PM  470 data  240,  31, 160,   0, 177,  95, 200,  81
FC  480 data   95,  16, 228, 177,  95,  16, 224, 160
DG  490 data    4, 177,  95, 197, 251, 144, 217, 197
JA  500 data  252, 176, 212,  32, 170, 203, 144, 208
NF  510 data   96,  24,  96,  32,  83, 203, 176, 249
MA  520 data  160,   2, 177,  95, 197, 251, 144,  10
NJ  530 data  197, 252, 176,   6,  32, 170, 203, 144
CI  540 data    2,  96,  24, 169,   3, 101,  95, 133
DJ  550 data   95, 144,   2, 230,  96, 197, 253, 208
```

```
KJ | 560 data  223, 228,  96, 208, 219, 240, 212,  24
CL | 570 data  165, 253, 133,  95, 134,  96,  69,  49
JE | 580 data  208,   4, 228,  50, 240,  39, 160,   2
LH | 590 data  177,  95, 101,  95, 133, 253, 200, 177
AH | 600 data   95, 101,  96, 170, 160,   0, 177,  95
KG | 610 data  200,  81,  95,  16, 218, 160,   4, 177
HL | 620 data   95,  10, 105,   5, 101,  95, 133,  95
MJ | 630 data  144,   3, 230,  96,  24,  96, 165, 248
EA | 640 data  133,  79, 165, 249, 133,  89, 160,   0
OM | 650 data  132,  78, 132,  88, 166, 250, 232, 177
EE | 660 data   78, 145,  88, 200, 208, 249, 230,  89
IF | 670 data  230,  79, 202, 208, 242,  96,  72, 120
JD | 680 data  169,  53, 133,   1, 104, 197, 248, 144
KM | 690 data    5, 229, 248,  24, 101, 249, 133,  79
DI | 700 data  136, 177,  95, 133,  78, 136,  56, 165
CO | 710 data   51, 241,  95, 133,  51, 200, 145,  95
PO | 720 data  165,  52, 233,   0, 133,  52, 200, 145
HG | 730 data   95, 136, 136, 177,  95, 240,   9, 168
OD | 740 data  136, 177,  78, 145,  51, 152, 208, 248
ND | 750 data  169,  54, 133,   1,  88,  24, 165, 254
IH | 760 data  229,  52,  96,   0,   0,   0,   0,  67
JG | 770 data   49,  57,  56,  53,  32,  70,  46,  83
DL | 780 data   73,  77,  79,  78
```

Some C128 Bits **Perry Shultz, Miami, Florida**

Ornament and Happy New Year In High–Res

```
9 graphic1:scnclr:color1,5:foru = 1to50step3:circle1,160,
  75,u,60–u:next:color1,2:forr = 9to85step5:circle1,160,
  r/9,r*2,r*3,,,,72:nextr:char1,13,18," happy new year ",1
```

Notes: The line number must be 9 or less. Type line with no spaces. After entering the last character, cursor back anywhere in the line then return.

Multiple Circle, Triangle and Square High Res Draw Routine

```
5 graphic1,1:fori = 25to300step9:circle1,i,100,20,
  18,,,,120:next:fori = 25to300step9:circle1,i,20,20,
  18,,,,45:next:fori = 25to300step9:circle1,i,175,
  20,,,,,90:next
```

Incredible 3–D Effect High Res Draw Routine

```
10 graphic1:scnclr:forr = 3to100step6:circle1,160,130,
   r + 20,r + 18,,,,120:nextr
15 graphic1:scnclr:forr = 3to100step4:circle1,r + 100,
   130,r + 20,r + 18,,,,120:nextr
20 graphic1,0:scnclr:forr = 3to100step4:circle1,160,
   110,r + 20,r + 18,,,,100:nextr
25 graphic1,0:scnclr:forr = 3to100step4:circle1,99 + r,
   110,r + 20,r + 18,,,,100:nextr
30 graphic1,0:scnclr:forr = 3to100step4:circle1,160,
   110,r + 20,r + 18,,,,90:nextr
35 graphic1,0:scnclr:forr = 3to120step3:circle1,r + 70,
   r + 20,r + 20,r + 18,,,,90:nextr
40 graphic1,0:scnclr:forr = 3to100step4:circle1,160,
   110,r + 20,r + 18,,,,150:nextr
45 graphic1,0:scnclr:forr = 3to120step3:circle1,r + 75,
   99,r + 20,r + 18,,,,30:nextr
50 graphic1,0:scnclr:forr = 3to120step3:circle1,r + 100,
   95,100,r + 10,,,,75:nextr
55 graphic1,0:scnclr:forr = 7to100step2:circle1,160,
   r + 60,r + 55,r + 3,,,,72:nextr
```

More Ideas

Redefine two function keys as graphic 0 (textscreen), graphic 1 (hi–res screen) — this enables screen change with one key-touch.

With the 160 bytes per line, I hope to see many new exciting 1 liners.

And For the First Time. . .

Some Amiga Bits and Pieces

Got an Amiga? Then no doubt you learn something new nearly every day – we do! We at The Transactor would be most pleased if we could share your discoveries with all our readers. Same deal as "Bits" – we'll credit you and send a free one–year's subscription.

Some Notes About CLI

CLI, Amiga's Command Line Interface, is your interface to AmigaDOS. You can access CLI by clicking its icon on your WorkBench disk – the CLI icon appears if the "CLI on" option is chosen in "Preferences". When a DOS command is entered, the system looks for the command in the current directory, and if not found, in the subdirectory C on the SYS: disk (the disk that was booted with). See the article in this issue for a brief description of the DOS commands.

The disk–oriented nature of the DOS commands makes for a flexible system, since you can add and change commands at will. With a single drive though, it can be a problem doing operations with a disk other than SYS: (the one in the drive). For example, if you wish to get a directory of another disk, you can't just switch disks and type DIR because the system will ask for the SYS: disk again (by volume name) and then do a DIR, giving you the directory of your original disk. Since AmigaDOS is a fairly flexible and powerful system, there are many ways of getting around the problem; here are a few suggestions:

1) The standard method is to refer to the new disk by name when giving the DOS command, for example to get a directory

of a disk called "Utilities", you could just enter:

```
dir utilities:
```

The system would then put up a requester asking you to insert volume "utilities" in the drive, and would give you a directory after you had done so. You can work with any file or directory on the new disk in this way, for example:

```
type utilities:stuff/TextFile
```

. . .would display the file "TextFile" in the sub-directory "stuff" on the disk "utilities". This method works fine when you know the volume name of the disk you're interested in (which you should, since you've thoughtfully written it on the disk label, right?), and you only want to use the disk a few times and don't mind swapping disks back and forth.

2) If you wish to switch to a new disk for awhile to perform several commands, and the new disk has those commands on it (usually in the C sub-directory), you can just change the assignment of C: , telling the system to look elsewhere for commands. For example, if from the original disk you typed:

```
assign c: utilities:c
```

You would be prompted to insert volume "utilities:", and the C sub-directory on that disk would then be searched for all DOS commands subsequently issued.

Likewise, you could re-assign the current directory using the CD command, as in:

```
cd utilities:c
```

The disadvantage with this approach is that it locks you into C as the current directory.

3) A more direct approach for using a new disk which also contains the DOS commands is to refer to the disk explicitly when issuing the command, preventing DOS from requesting the SYS: disk. For example, if you wanted a directory of any old disk laying around (remember, it MUST contain the required DOS command – in this case DIR – in the C directory), just pop in the new disk and type:

```
df0:c/dir
```

That way you are referring to Drive 0 (not a specific volume), C directory, then finally the command name. This is a handy technique for little one-time commands such as a DIR or TYPE when you don't feel like typing in or don't know the new disk's volume name.

4) A favourite trick used by many is COPYing all or some of the DOS commands into RAM and then assigning C: to RAM to tell the system to look there for the commands. You could use the following sequence of commands, possibly in your startup-sequence batch file, to accomplish this:

```
makedir ram:c  ;make c sub-directory in RAM:
copy c: ram:c  ;copy entire c sub-directory to RAM:
assign c: ram: ;assign ram as new source of commands
```

This seems to be the ultimate solution at first glance, since all of your commands execute out of RAM at lightning speed, and you're never bound to a disk when issuing a command. The disadvantage (there had to be one) is that you use up lots of RAM, and also (OK, two) it takes a long time to copy all of those commands. Nonetheless, some people have enough RAM and enough time that this really **is** the ultimate solution to fast and flexible DOS commands.

5) A variation on the above RAM technique is my favorite, thought up by Amiga-buff Rico Mariani. Pick your most-used DOS commands, for example DIR, LIST, COPY, ASSIGN, CD, and TYPE, and copy them to RAM. Then assign names to each of those files, and use those new names in lieu of the command names. (ASSIGN is just a way of setting up a new name to refer to a volume, directory, or file.) As a confusion-avoiding convention, make the assigned names identical to the command names, except for the required colon (:) at the end. The example below should clear up any confusion (you could use this in your startup-sequence).

```
copy :c/dir to ram:
copy :c/copy to ram:
copy :c/cd to ram:
copy :c/type to ram:
assign dir: ram:dir
assign copy: ram:copy
assign cd: ram:cd
assign type: ram:type
```

Now, with those assignments in place, when you wish to do a DIR, just type dir: (with the colon at the end). This will get the dir command from RAM, executing it quickly, and you don't have to have the dir command on the disk currently in the drive. Also, you haven't use up tons of RAM, since you've only copied the commands you need. Obviously the assignments aren't needed at all, since you could just use "ram:dir" for the same effect, but the assignments make things just a bit clearer and easier to type. Incidentally, you can use assign whenever you'd like to use an alias to refer to a directory or file. Tired of typing "execute" all the time? Just do an:

```
assign !: c/execute
```

and use !: instead of the word "execute" at any time. Assigns are system-wide, not just for the current window, so your assignments will last until re-boot (and beyond, if you put them in the startup-sequence).

Letters

1200 BPS Response: Reference 'Twinkle Tones' in your 'Letters' column of Transactor, Jan. 1986: Volume 6, Issue 04. Mr. Giese stated that "One of the things that I have learned in playing with my C–64 is the amount of mis–information available!". How true. But he then goes on and spreads more mis–information about the C64 in the remainder of his letter!!

Mr. Giese implies (states?) that you can't use a 1200 bps modem with the C64 until you do something special. This is just not true!! I have been running 1200 bps modems with my C64 for two years now with several different ternimal programs and have had no trouble. And I didn't have to do anything special to run these 1200 bps modems – THEY DO WORK WITH THE C64!!

Mr. Giese then states that in opening the RS–232 channel, setting the control register to CHR$(8) and the command register to CHR$(0) does not work for 1200 bps, one stop bit, 8 bit word, no parity, and full duplex. He further states it is because the baud rate table in the Programmer's Reference Guide is wrong (that is how the CHR$(8) was determined). That is simply not correct, as I have been using the same baud rate table for a number of different baud rates and it does work!!! I used an old RS–232 dot matrix printer at 300 baud with my C64 for about a year by using the baud rate table for a number of different baud rate inputs. I have successfully printed on the printer using 300, 1200, and 2400 baud by consulting the baud rate table and setting the control register to CHR$(6), CHR$(8), and CHR$(10) respectively, and setting the command register to CHR$(0)!!! I am having a buffer problem at 1200 and 2400 baud which is due to handshake and/or cable inconsistencies, or possibly the lack of a suitable buffer in the printer. But the fact remains that you can communicate at 300, 1200, and even 2400 baud from the RS–232 port using the baud rate table as published. I do it!!!! Mr. Giese's problem must be something else, not the baud rate table.

Lastly, Mr. Giese states that the Programmer's Reference Manual infers that the User baud rate is not implemented. Pages 349 and 350 of my Programmer's Reference Guide infers no such thing – in fact it infers the opposite! It tells you how to calculate a user defined baud rate, but the calculations seem to have at least one error in them.

Mr. Evers, I hope that your magazine will clear up this additional mis–information about the C64 and the RS–232 port that was published in your Jan. 1986 Transactor magazine (the first issue I have ever read).

Albert F. Harsch, North Huntington, PA

Now we're really confused. You say it works, Lyle Giese says no. Actually it was Rick Sterling and Joe O'Hara of Microtechnic Solutions that supplied Mr. Giese with his information. However, you must agree that three users **would** *have trouble experiencing a problem that doesn't exist. Perhaps the next letter will help shed some more light in this dark area of 'inner space'.*

More Responding at 1200 BPS: In Volume 6, Issue 4 (January 1986) of The Transactor you published a letter from Lyle R. Giese, Woodstock, Illinois, called "Twinkle Tones". Among other things RS232 baud rates as implemented on the C64 were discussed. I believe I may have something useful to add to the discussion.

Mr. Giese rightly points out that user–definable baud rates are implemented by the C64 Kernel. Confusion may arise on this point because I believe that the Vic 20 Kernel does not implement user–definable baud rates – or rather, it does, but in an incorrect manner. To understand what is going on requires a little technical detail. I hope to make it fairly clear in what follows.

What happens in the C64 Kernal when an RS232 file is opened is this: the OPEN command is followed by a filename field of one to four characters. The low nybble of the first character is used as an index into a baud–rate table in the kernel (one of two separate tables is used depending on whether system frequency is NTSC or PAL). The values in the baud–rate tables are pre–scaler values for the CIA #2 Timers A and B, which are used to time the non–maskable interrupts of the RS232 rou-

tines (Timer A is the transmit clock and Timer B is the receive clock). More accurately, the pre-scaler values in the baud-rate tables are what might be called the half-bit times. The values are half the number of clock cycles it takes to transmit or receive one bit at the selected baud rate. The OPEN routine uses these values to calculate the full-bit times, and stores both values in page 2 locations in lo byte/hi byte form (the half-bit time is at $0295/96, and the full-bit time is at $0299/9A). The RS232 routines in the kernel make use of both values, although exactly how they do so is beyond the scope of this letter.

We are now in a position to understand exactly how user-defined rates are implemented on the C64. If the low nybble of the first character in the filename field of the OPEN command is zero, then the pre-scaler values are not obtained from the baud-rate tables at all. Instead, the third and fourth characters of the filename field are considered to be the pre-scaler values. This is exactly what the formulas in the Programmer's Reference Guide produce - the half-bit time in lo byte/hi byte form.

We can also begin to appreciate where some of the confusion surrounding the RS232 routines in the kernel has arisen. In the first place, the low nybble of the first character can have sixteen different values, but not all sixteen mean something. Zero gives the user-defined rate, and there are ten different values in the baud-rate tables. What happens if the low nybble has a value greater than ten? Simple - there is no error checking, so the index now points to random bytes beyond the end of the baud-rate tables - and those are used as the pre-scaler values. Why doesn't the user-defined rate work on the Vic 20? As I understand it, the kernel of the Vic 20 does not specifically check for a zero in the low nybble of the first character - now the index points to somewhere before the beginning of the baud-rate tables, and again a random value is used for the pre-scaler value. This is apparently a bug which was fixed in the C64.

One more thing which is now apparent is why the C64 has trouble with some 1200 bps modems. By examing the baud-rate tables it is clear that the baud rate produced when 1200 bps is requested is --- exactly 1200 bps. What can possibly be the problem, then? It turns out that it is the 1200 modems that do not operate at 1200 bps. Typically, 1200 bps modems actually transmit at 1219 bps and receive at 1182. Some modems are more tolerant of deviations from these rates than others, particularly as regards the receive rate (which is good, since the RS232 routines in the C64 kernel do not operate at different transmit/receive rates, although there is nothing in principle to stop them - it is mostly a question of obtaining separate timing values for the two clocks. A fifth and sixth character could be added to the OPEN filename field for a differing transmit rate, perhaps, along with space to store them and routines that can find them). The pre-scaler values given by Mr. Giese in his letter (CHR$(57)+CHR$(1)) actually works out to about a 1238 bps rate. For myself I have found that CHR$(64)+CHR$(1) works well. The actual value found in the NTSC baud-rate table corresponds to CHR$(70)+CHR$(1).

I hope this helps clear up some of the mysteries surrounding the C64 RS232 routines.

Anton Treuenfels, Fridley, Minnesota

Almost Clear: While reading one of my Transactors, Vol. 6, Issue 01, I came across a thing for clearing a line on the screen. In the issue before there was a letter saying that a guy had a program in which the top 3 lines must remain. The editor gave a long IRQ routine to do this. I have devised a way to do the same thing using the clear screen line technique. Here is an example of how it works:

```
10 for p=1 to 38: print "abcdefghighlmnopqrstuvwxyz";
   : next
20 for l=1 to 1000: next
30 for k=3 to 24: poke 781,k: sys59903: next
40 for l=1 to 1000: next
```

The program fills the screen with letters. There is a pause then all the lines are cleared except the top 3. Then there is another pause allowing time to show the results. The line in which you would want to use and change to your own needs is line 30. This line does all the work. I hope that this will relieve the use of the long IRQ routine.

Mike Digdon, Bedford, Nova Scotia

Thanks for the code. Line 30 measures in at 30 bytes of Basic (without spaces), no assembler required. As far as speed is concerned, it goes off pretty darn quick. It's nice to see a better cure for a problem.

Incidently, the IRQ driven routine in Volume 5, Issue 06 was a continuous screen display routine for the top three lines. It measured in at 43 bytes of object. The partial screen clear routine was written to be called as required, with a total object count of 25 bytes. Although written in assembler, they both were not too terribly long. Now, to take your routine and re-write it in assembler, we come up with this:

```
*       =    828     ;cassette buffer
;
        ldx  #24
;
loop    =    *
        jsr  59903   ;clear line specified in .x register
        dex
        cpx  #2      ;are we done yet
        bne  loop    ;nope!
        rts
```

Pronto! And only 11 bytes of object. Not too shoddy. Thanks for the idea.

Transactor/Ohio Porting: Just a note with my subscription to tell you that I am enjoying your magazine immensely. Since discovering The Transactor, I buy it before any other on the newsstand. I do not currently program on any Commodore!

Your articles are great for programming ideas. The Jan '86 issue, Vol 6, Issue 04 particularily impressed. The SID super sound commands have been implemented with my OHIO SCIENTIFIC BASIC. Your article on the SID filters is very informative. The projectile motion article sits in my mind as one to try to implement on the Ohio.

While I realize it is not always possible to publish source code due to the length of the code, I would appreciate it being available as often as possible. Perhaps information could be included as to contacting you or the author for the missing source. As you know, this would simplify my task, and give me many ideas.

Another issue that I enjoyed was the one dealing with communications, particularily the serial bus information pertaining to the Commodores, but applicable to many other situations.

You may well know that the Ohio Scientific release of Microsoft Basic is dated just prior to the first Pet, and thus has much in common. I currently use a mostly C4P model, with 48k of RAM. Beside the 64X25 video, I also have a T19114 video controller chip implemented. This gives 16 colours, Hi-Res with Apple compatible plotting commands, with a resolution of 255x192 pixels. It also supports 32 sprites. I added a SID chip for sound and a 6522 to drive a parallel printer. I expect to put a GI AY3-8910 into the ext pin of the SID.

Not bad for a system produced at the same time as the original Pet. The one big advantage I have is that the Basic and the operating system are loaded into RAM from disk whenever the system is booted. It thus lends itself admirably to tinkering.

John Horemans, Mississauga, Ontario

You could take the prize for the greatest amount of perserverance in recorded history. While the rest of the world is scrambling for the newest, biggest, bestest, fastest, greatest, most incredible, you quietly work with your loyal friend, modifying as desired. Anyone who reads the Transactor just for the ideas has to be special. Thanks for a refreshing letter. Generally we print source code with few exceptions. Even long listings aren't too unreasonable once they're reduced with Karl's "typemagic". But in this case we didn't get source code from Mr. Reesor (4408 63rd St, Camrose, Alta, T4V 2J4). Perhaps he can help, otherwise you may need to use "Unassembler" (Disk 9, same as Super Sound) to make source on a friend's Commodore.

A BIT Of A Problem: Congratulations to Transactor and J. Lothian for the excellant "Disk Un-Assembler for the Commodore 64" in The Transactor (Volume 6, Issue 04). There is a BIT of a problem, but it is a superb piece of Basic code.

The BIT operation is a problem as Lothian hints, and presumably space prevented further elaboration. But this BIT problem is the most likely source of a crash of the un-assembler. For example, if you use a monitor to disassemble the standard C-64 disk wedge, at $CE2F you will find the unusual syntax of BIT $00A9. Only BIT $A9 in proper zero page syntax was necessary, and presumably an assembler gremlin stuck in the leading zero. The microprocessor executes op code $24 (BIT) as zero page mode, and op code $2C (also a BIT) in absolute mode. The first uses 2 consecutive bytes, and the latter requires 3 consecutive bytes. As far as the executed result is concerned, BIT($24) $A9 and BIT($2C) $00A9 are the same. The latter simply takes up a bit more memory, and executes a bit slower.

But "The Commodore 64 Macro Assembler Development System" doesn't treat these two syntaxes the same. If that assembler encounters BIT($2C) $00A9 in the source code, it presumes that you made a mistake and automatically drops the leading zero, converting it to BIT($24) $A9. The result of an un-assembly and re-assembly of such code is that a byte (the zero) is dropped, and all of the following code is offset by one byte. Naturally that leads to a crash.

So, I encourage inclusion of Lothian's suggested line 61 in the program that causes the un-assembler to burp out a .BYTE, as if BIT didn't exist. The Commodore assembler then fails to drop any bytes from the source code! Although a few lines of code thereafter may appear incorrect in the source listing, the assembled result will operate correctly; and it's fine as far as the microprocessor is concerned, which is the main thing, after all.

I note that lines 2020-2060 are never accessed in the un-aasembler program, which is a shame because they would signal that a BIT operation was converted to a .BYTE. Also, lines 310-360 in the program are excess baggage, although it would be nicer if that weren't the case. Presumably Lothian tried to stick closely to Higginbottom's prior program code, while disagreeing with his treatment of BIT. I agree with Higginbottom that BIT must be converted to .BYTE in any un-aasembler which is related to the standard Commodore 64 assembler.

That might not be true with other assemblers, which may have entirely different quirks. I don't really know. I can only afford one commercial assembler. Because I bought the Commodore assembler fairly early on, that will have to suffice.

Another problem with the un-assembler is that it won't create more than one file, despite what is claimed. I suspect that a couple of variables got mixed up, nevertheless there is an easy cure without striving for elusive perfection. Substitute the

following line in the program:

1280 IF LC<1000 THEN 1350

That fixes everything except when you try to un-assemble a long bit (there's that gremlin again!?) of machine code. The un-assembler inserts a lot of unnecessary spaces in the source code, creating a significant limit on the amount of source code which can be created on ordinary diskettes. With an ordinary 170k 1541 diskette, only about 12k of machine code can be un-assembled. Deleting unnecessary spaces from the generated code can double the limit. 24k sounds and is a lot better! This change is fairly easily accomplished by going through the program and substituting a single space wherever there are multiple spaces, notably in lines 1250, 1260, 1380, and 1400. It also seems desirable to increase the dimensions in line 120 to L1(2000) and L2(2000), which will then handle up to about 24k of machine code in a single un-assembly. (at least, after compilation)

But let's not get carried away with deleting spaces! The proportional typesetting machine used to set Transactor program listings is a problem too. Be sure to include a space between .BYTE and $ in line 1480. Otherwise the Commodore assembler generates the " RAN OFF END OF CARD " error message. That error message presumably means that a couple of cooties sitting on the Ace of Spades will never really know whether the card is flat or round.

Finally, after the corrections and changes as indicated above, I want to confirm that the un-assembler works very well. It can be easily compiled with the Abacus compiler to give a 3-4x increase in overall speed, with negligible expansion of the program code. My compiled version of the un-assembler took about 4 hours to un-assemble 21k of code, i.e. almost the full capacity of 1 170k 1541 diskette. After deleting unnecessary spaces from the generated code, the resulting code is approximately 5-6x expansive.

Incidently, I confirmed that after the BIT to .BYTE correction, the un-assembler correctly handles the standard C-64 disk wedge program, allowing relocation of the utility to any memory area simply by varying the first line in the source code. Relocation of machine code, as much as editing, is a major advantage of the un-assembler.

I really enjoy Transactor, at least partly because you obviously do too. John R. Menke, Mt. Vernon, IL

There are quite a few extra benefits derived from working at The Transactor; one of them is the continuous stream of top notch letters and articles originating with John Menke. It's always a pleasure to be on the receiving end of your thoughts and observations. Your comments, as usual, are A1. We thank you for making what would have been just a good program – great! Please keep the correspondence coming.

Left Wing Interference: I had an experience this weekend that I thought might be of interest to other users of the Commodore 1541 disk drive.

My son's "Winnie The Pooh In The Hundred Acre Woods" program was having difficulty loading some of the screen files, and would sometimes provide an error message indicating a problem with the disk drive. This led me to believe that the drive might be out of alignment. So I checked with the "Check/Adjust/Alignment" function of the "1541 Disk Drive Alignment Program" from CSM Software. This function determines the time to access every seventh sector of every fourth track of a calibration disk supplied with the program. Proper alignment is indicated if the program reports a 'timing number' of about 100. The program was indicating timing numbers of 110 to 113, and blinking of the red light on the drive indicated that there was difficulty in accessing sector 8 of tracks 5 and 9.

The disk drive and TV normally sit on the top shelf of a cart wich I roll up to a side arm of my desk, where my Commodore 64 is set for use. Because there is not enought room on the cart to disassemble and adjust the drive, I moved it to my desk top. I then rechecked my timing number and found that it was 101 to 102 – well within the acceptable range – and there was practically no trouble accessing the disk. However, upon returning the drive to the cart, the timing number returned to 110 or greater.

A little investigation showed that if the disk drive was sitting to the left of my TV (or my Commodore 1701 monitor), there was trouble accessing the drive. When the drive was sitting to the right, there was little or no trouble.

In conclusion, sitting the disk drive to the left of a TV or monitor can produce symptoms which mimic alignment problems. Readers might want to check for this type of interference before going to the trouble of having a drive realigned.

Jack Ryan, El Dorado, Arkansas

A while ago I received a 1541 fast load cartridge called GT-4 from Proline for review. The fast load was interesting, but what was more enlightening was the manual supplied. By reading through the authors notes, a similar experience to yours was noted. The author wrote that odd gremlins appeared within the 1541 if operated too close to the left side of the Commodore 1701/1702 monitor. Specifically, trouble might occur reading from and/or writing to track 35. Through your own experiences it seems that the problems are further reaching than just track 35. Very odd.

Perhaps, and this is pure and applied speculation, the trouble lies not with an actual read/write error, but with a checksum error in the data read/written. The flyback is placed closer to the left side of the Commodore monitors. Perhaps operating the 1541 too close to the flyback is sufficient to cause bit movement at the head or some other unexpected spot within the drive.

Although the diskette is actually in good shape, and the alignment is Ok, the checksum always bombs out thereby flagging an error. A theory worth considering. It's worth mentioning here that I originally shot off on an altogether different tangent blaming speed variations due to the physical placement to the monitor, but my father brought me back down to earth. Thanks Dad.

Here is another tip gleaned through exposure to my father. Once again; thanks Dad.

In many cases of supposed alignment problems with the 1541, the root of all evil can be found in the form of a speed error. Speed variations can be caused by a variety of reasons, one of which is not using it for an extended period of time. If this sabbatical is spent basking in a fairly warm to hot environment ('School's Out For Summer!'), the demon may appear. Given these conditions, the drive belt will dry out thus taking on the shape it is currently in, an oval. Once the drive is fired back up again, the speed will be all over the place due to the rigid malformation of the belt. The cure in this case is to either replace the belt or continue using the drive until the belt loses some of its rigidity, or consider just popping the belt off and leaving it inside the case if extended non-use is anticipated.

The Gremlin Effect: It seemed to me that your current piece about errors on page 14 of Vol 6, #04 could apply to my recent letter of last June 12th.

In it I complained that two programs from the July, Vol 6, #01 Transactor just wouldn't work for me and in fact kept producing endless loops and fouled up generally! Namely, your own " File Pursuit " and Jeff Goebel's " Bootmaker II " .

I decided to have another crack at them tonight. Taking a brand new disk, loading " Verifizer " into my 64, I re–entered " File Pursuit " and this time it worked absolutely perfectly!

I went back to my orginal "save" on an older disk and re–loaded that for comparison. Apart from a few spacing differences between some words or commands, the two versions appeared to be identical when listed on screen in manageable groups of lines. Yet running the earlier one produced exactly the same hang–ups, endless loops, etc!!! Even re–entering the suspect lines several times had absolutely no effect! Just as though the program was jinxed from the word 'Go'!

Needless to say I replaced the first one on the old disk with my new workable one, as a back–up copy and gave the working " File Pursuit " a place on my main 'Disk Utilities' disk. I then re–entered " Bootmaker II " and got that working first time too! I had destroyed my earlier "save", so couldn't compare the earlier one that had destroyed a ml program!

Quite frankly I had begun to think that my 64, which may well have an earlier ROM chip in it, had a few weird bugs inside! I know when I foul up entering lines, though I've often found that even if one spots an error and cursors up and re–works the line, it may never be right. In which case one has to redo the entire line. I have had this happen with much longer programs in other magazines and books. Finally ALL the major Commodore 64 magazines now have checksum type "goof–proof" programs, or at least ones which make it pretty hard not to catch errors right away. However a series 'RUN' ran on a 'Basic 4' by a Canadian author, I still can't get to work at all and I've had file loading problems with an English book on ML utilities.

Reading 'Transactor' has persuaded me to lay out money on books on machine language and 1541 DOS, so this winter I should be all set to grow some more computer–wise.

Quite frankly I'm beginning to think that my 64 does have a resident *gremlin* inside, one that from time to time had a decidedly " off " night and refuses to allow a program to be entered correctly. If I hadn't experienced this with " File Pursuit " and " Bootmaker II " , I'm not sure I would have believed it could happen!!!

Looks like I'm going to have to get back to re–working a few more programs now! At least while the gremlin is in a good mood!

John Matthew, Rexdale, Ontario

Although Verfizer will catch entry errors, it ignores Spaces. About the only critical Spaces possible in CBM Basic are those in Block Commands sent to the Disk Command Channel. However, we used semi-colons in the Block Commands of File Pursuit for just this reason. So it certainly is mystifying that two identical entries would not work the same. Try loading Verifizer and your first "save" of File Pursuit and just hit return over each line while checking the Verifizer codes. This may lead you to the discrepency, which would certainly be interesting if not educational.

Jordan Rolltop Stand Revisited: I am writing concerning the "Jordan Rolltop Stand" in the Transactor, Volume 6, Issue 05. First from a technical standpoint; having grown up in the lumber business and having built much of the furniture in my house, I would like to say that the article was well done and the assembly instructions easy to comprehend, although I did notice that the stand in the accompanying photo was not made to the specs in the article. Also, I would caution (indeed, I would PREACH!) against buying lumber for a project like this at a regular lumber yard because building supply yards generally carry only lumber for making buildings. This is not acceptable for furniture type projects. Here's why; building lumber is usually kiln dried to approximately 16 to 20 percent moisture content (m.c.) but to match the humidity inside a house, lumber must be drier; around 4 to 6 per m.c.

You can build with construction lumber fine, but as it sits in your house - which is probably about 6 to 8% m.c. - the lumber will dry further (to "equalize" itself to its surroundings). When this happens, the piece will shrink and/or warp slightly, sometimes causing real problems, especially in furniture with moving parts. For a piece of furniture meant to last, lumber dried to 6 to 8% m.c. is a must.

But my point in writing is not to critique the article, but rather to offer help to Transactor readers. To anyone who is inclined to build the "Jordan Rolltop Stand" (or any woodworking project for that matter) we can supply quality Appalachian hardwoods and New England White Pine in any form from rough sized, surfaced boards right through to "ready-to-assemble" pieces, and can supply them for less money than most furniture lumber suppliers. Take for instance the "Jordan Rolltop Stand"; if you wanted to make this out of Cherry or Hard Maple you could get the rough sized boards to do it for about ten dollars (U.S.) (depending on the actual size of the desk it was to be placed on) or get ready-to-assemble pieces, less the cloth for the tambour, for about twenty-five dollars (U.S.). Shipping charges are extra and will vary depending on distance.

If you've got your own idea for a project, send a drawing or picture – or a cutting list if you have one – along with a five dollar (U.S.) deposit and we will draw the plans for you and send you a custom price quote. If for some reason you don't buy the lumber, the five dollars covers our time in drawing up the plans. (No deposit is necessary if you send a cutting list.) Traditional furniture woods, such as Ash, Cherry, Hard Maple, Poplar, Red Oak and White Pine are all readily available and most other North American woods can be gotten (as long as you're not in a hurry!) So don't decide not to build that project because of fear of saws, lack of ability, or any other excuse you may have for leaving your computer area in a mess; we'll meet you right where you want with the lumber you need.

Matthew Strange
P.O. Box 2
Mansfield, PA
USA 16933

First, let me quote from a hand-written letter attached to this one when we received it:

" If this letter sounds too much like blatant advertising - feel free to toss it. I've been considering starting up a business like this for about 2-3 years now and this looks like a good place to check out its feasibility and help out my computer friends as well. But like I said, if it's too blatant in its advertising content - chuck it out. "

Terrific! Anyone who asks us to "Chuck It" if we begin to feel compromised can't be too bad. Here's wishing you massive oodles of luck with your business. I like your style.

The Horror Of Hex: I have hesitated writing this letter for over a year now but, even though I realize it is like trying to prevent the sea from following the moon around, I am going to try to have my say.

I own a Commodore 64. I am quite familiar with it and I can program fairly well in M.L. My frustration comes from bumping into HEX notation all the time. It is utter nonsense.

The use of Hex is a game some programmers play. My computer does not understand Hex. When I Poke a value into memory, both the memory and the value must be in decimal!

I have a memory map of low and high ROM's with each memory location given in Hex. What a waste. I had to translate every hex address into decimal before I was able to peek the routine or before I could SYS to it. The entire process of figuring the HEX code to prepare the map was a waste of time and caused untold hours upon hours of wasted time as programmers everywhere decode the Hex back into useable form.

When I read an article in a magazine (Such as Transactor), I have little if any difficulty following and understanding the flow. 90% of my time is wasted in flipping back and forth through my Hex to decimal conversion chart!

If you really take an objective viewpoint, you will also see how wasteful HEX is. Even Transactor publishes all 'data' statements in decimal. THE LINE NUMBERS IN A M.L. LISTING ARE IN DECIMAL!!!

If HEX is really so great, why don't you use it for the line numbers and for the PAGE NUMBERS! Frankly, I think HEX is to programmers what Latin is to Doctors. It helps support a private clique to which the "undesirables" cannot belong. The Doctors are gradually giving up Latin and I would be thrilled if programmers would give up HEX.

In doing let me say I do understand HEX - I just see no need for it in any home or business computer environment.

There - I said it.

Thomas W. Gurley, Willis Point, Texas

Time is the major factor involved in learning to appreciate the hexadecimal numbering system. Similar to olives, artichokes, smelly cheese, or whatever initially disgusting ingestible, Hex requires that you learn to accept its obvious rude points before you are allowed to enjoy it.

You may have noticed that new BASICs are including HEX and DEC functions to allow conversion for those who wish to SYS, POKE, etc., to an address specified in hex. Although you do have a valid point for the hybrid BASIC/Machine Language environment, the fact remains that hexadecimal is the only notation for the total Machine Language situation. I suppose if

ICs where originally designed with, say, 6 address lines (0 to 999999) and 3 data lines (0 to 999) then decimal would have fit much more naturally. Except each line of every chip would need the capability to sense 10 voltage levels and that might be expensive. Besides that, "I Adore My Commodore 100" doesn't rhyme and a "K" would actually be 1000 bytes – now that's confusing!

You might say that binary would then be the most natural since there are only 2 voltage levels, but groups of 8 and 16 '1's and '0's take up far too much paper space. Hex merely allows you to "see" four of those characters by only looking at one. I guess, once again, it's something you get used to in time. I know some programmers that swear by Octal!

False ID: The Transactor is a real winner and I always look forward to a new issue. You have published many good articles on the 1541 disk drive. However, I cannot find anything that tells me how to change a disk ID without destroying what is on the disk. I know the ID is at byte 162-163 of track 18, sector 0, and I have no trouble changing that, but I understand the ID is printed to every sector on disk. When I display other tracks I do not get the ID information. I would really appreciate your help in this matter.

E.C. McPherson, Ottawa, Ontario

Commodore DOS is very unique in concept; during a formatting procedure, it magically tucks away the diskette's ID with every sector. The magic part of this entire procedure is that the average user would never know of its existence.

The DOS uses this ID as a check-sum to ensure that the diskette is Ok while you work with it. If for some reason a sector gets messed up, the ID offers DOS another method to detect an error. Along with this hidden ID are a whole slew of other equally important bits of information. There is only one problem: Commodore wrote the DOS to make this portion of the sector difficult if not impossible to access. They felt that there was no reason why anyone would ever want to dig this deep into their operations. Therefore, you are going to have problems changing the ID.

Considering that you did not mention why you need to change the diskette's ID, I will assume that either you are just nuts about arcane bits of Commodore tech, or you are brewing up some form of disk protection for a package. Whatever the story, this subject is a bit too involved to cover in the letters column, and we suggest you don't try changing any occurrence of the ID unless you're prepared to do a complete job on the entire disk. If that's the case, try locating the book "Inside Commodore DOS", written by Gerald Neufeld and Richard Immers. You would be hard-pressed to find any other book that would compare to this one for arcane bits of information about Commodore DOS. These guys rank right up there with Raeto Collin West of "Programming The PET/CBM" fame in the presentation of high-level information. Try hitting a few of your local Commodore dealers for the book. There is a good chance that if they don't have it, they would at least know of it and point you in the right direction.

Attack Of The Killer Clone: I have an INDUS-GT disk drive and I have problems. I cannot find any information for this drive except for the manual(?) and instructions(?) that came with the unit. Where can I obtain more information? There has been nothing in any Commodore magazine about this drive, except advertising this drive for sale.

I am the President of TRACE (The Richmond Area Commodore Enthusiasts), with 68 members. There are a dozen INDUS-GT's in the group, and each have problems.

We cannot get Microprose software to load on the INDUS-GT or MSD-1. All calls to Microprose have been less than satisfactory. Only tested on the 1541 and not guaranteed to work with anything else.... What goes? All other software written and on disk from other companies Load and Run fine on INDUS and MSD-1. Programs Saved on disk to the 1541 and Loaded into INDUS or MSD-1 work.

We have tried everything but nothing works. Help!!!

A quick bit of computer trivia of days gone by might put this problem in perspective.

A long, long, very long time ago, the Abacus was born of two very good friends, Aba and Cus. The Abacus was conceived due to a difficult problem Aba and Cus encountered on a daily basis. In their village, Aba and Cus were considered to be the finest mathematicians within a 10 miles radius. Due to this fact the townspeople would often rely on the guys to straighten out whatever financial mess they found themselves in. Initially, this presented no problem. Ten fingers and ten toes were more than sufficient for even the most vexing of problems. But soon, as can be expected with any successful small business, their sum and total of anatomical parts were insufficient for their needs. Enter; the Abacus.

Aba and Cus build the first unit with absolute love and devotion. When complete, it was just perfect. Calculations that before would take many days to complete were performed on the Abacus within a period of a few minutes. And so, the legend of Abacus began. They sold out of their financial consultation firm and began the Abacus Manufacturing Company.

As could be expected, the larger Abacus got, the more problems they had with imitations coming onto the scene. The original clone was born, and with it came the inherent problem with most clones. You have to expect imperfections when you are dealing with a copy.

As time moved along, Abacus was deluged with requests for help regarding Abacus clones. Problems such as beads falling off, beads seizing in place, insufficient or too many beads supplied and, of course, no beads, were in great quantity. In performing a market survey, they found that 95% of the clones had very obvious shortcomings, with the balance presenting themselves as perfect replicas. But Aba and Cus had been in business for so much longer that even the sum total of all the clones added up to only a small percentage of the number of originals they sold. They found they could not possibly spend the time coverting their Abacusware to work on anything but their own units. Third party abacusware developers felt the same. It soon became the responsibility of the clone manufacturers to be more compatible. However, they decided not to since they couldn't possibly anticipate every exception outside "normal use".

The moral of this story, as I am sure Aba and Cus would agree, is that a clone may save you money, but you have to be prepared to live with the problems of using a copy. The INDUS-GT and MSD-1 drives are compatible with the 1541 in many respects. In truth, each have superior features to the 1541. But they are not absolutely compatible. If each ROM routine was the same, then Commodore would have a really good chance of winning a juicy little lawsuit. Both manufacturers did produce almost perfect clones, but the almost perfect is the killer. Disk protected software packages often rely on techniques of bypassing the DOS's Interface Processor and working directly with the Floppy Disk Controller. At this level of operation, anything is possible. A good chance exists that quite a few packages use the DOS ROM routines directly, along with utilizing little known quirks of the 1541.

My only advice, as far as software packages go, is to be careful and hope that either the software or hardware manufacturers are responsible enough to listen to legitimate complaints. Otherwise, be prepared to tear down those programs yourself to find and modify that one piece of non-portable code that stands in your way.

As far as documentation goes, try to get hold of whatever Commodore drive users manuals you can find. This and a few Commodore DOS books and articles will probably help you out as much as you can expect. If the drives are clones, Commodore documentation should suffice. Hope the strange advice helps.

Transbloopers

Hi-Res Terminally Ill: After reading "The Error Of Our Ways: More Often Oops Than Bloops" on page 14 of the Jan. 86 Transactor, I moved on to type in the HIRES Create program to use with the Projector program. It would not work, not because of a checksum error, but because of an out-of-data error!

I triple checked all the data statements with Verifizer and even counted all the lines. But this time it truly was your error.

Fortunately, I have a copy of The Transactor Vol. 5, Issue 06 where HIRES was first published. I compared the two listings and found that the first 30 data items are missing from the second listing. The number 51233 in line 1050 should be changed to 51231 and the checksum number in line 1060 should be changed from 245,919 to 245,727.

I also discovered that you had forgotten to write the starting address to the machine language program that HIRES created. So I added this line to Hires Create:

```
1045 print#8,chr$(0);chr$(192);
```

I ran the program again and then ran the Projector program and it worked beautifully. This is a great graphics program and I want to thank you for publishing it.

Tony Damato, Jacksonville, Florida

Strangely enough, it took quite a while before word started to filter in to us that HIRES was sick that issue. Sure enough, it was sick and it was our (my) fault.

To transform object into data statements for publication, I usually use one of two programs. The first is "Data Gen" written by Karl to transform any object into clean and neat Data statements. The second is the same with some heavy mods by yours truly. The heavy mods produce a Data loader that Opens a program file for a write to disk, takes the first two data elements as the start address and the balance as the object. From that point you have pure object on disk to be Loaded via ,8,1. The problem came when "Data Gen" ala me became terminally ill. It seemed to be fine, but it ate four of its own lines. Four lines x 8 elements per line = 32 data elements. Two elements were the start address, the balance was required code. The checksum was Ok, as was most everything else. All we can do at this stage is offer our apologies and hope that the corrections listed below help to make up for our mistake partially.

Missing lines from Gary Kiziak's Hi-RES routine re-published in Volume 6, Issue 04. Notice the start address is included as the first two elements of line 1082. Omit these if you plan to use line 1045 above.

```
1082 data   0, 192,  76, 194, 193,  76, 247, 195
1084 data  76,  98, 195,  76, 110, 194,  76,  30
1086 data 194,  76, 214, 196,  76, 228, 196,  76
1088 data  11, 197,  76,  67, 197,  76, 169, 192
```

TransBASIC Installment #8

Nick Sullivan
Scarborough, Ont.

The TransBASIC Disk

Since the TransBASIC modules published so far are consuming so much space on each Transactor Disk, we have decided to produce The TransBASIC Disk. Starting with this issue, only the modules published in each issue will be on The Transactor Disk. The others, plus several that have not yet been published, will reside on The TransBASIC Disk, which contains almost 500 blocks of source code. Included is a manual showing each command from every module with short examples. More complete documentation for a command can always be found in a Transactor back issue, should you need it. Otherwise you simply run "" and start adding commands*

After selecting commands from the library they need to be assembled with PAL. Until now! SYMASS 3.0 is a machine language assembler that is compatible with PAL format source code, and it will be on The TransBASIC Disk! (It will also be published in the next Transactor, with assembler-design theory and complete instructions) SYMASS 3.0 is not compatible with all of PAL's exotic features, and although it will assemble most any program, it is not a "development" package. SYMASS does not print listings and sends object code to memory only. So if you're writing code, PAL's error checking and elaborate pseudo-ops are still the ideal approach, but for assembling TransBASIC modules, SYMASS 3.0 is perfect!

The TransBASIC Disk with SYMASS 3.0 is just $9.95. See News BRK this issue for more, or use our postage paid order card at center page.

TransBASIC Parts 1 to 7 Summary:

Part 1: *The concept of TransBASIC – a custom command utility that allows one to choose from a library only those commands that are necessary for a particular task.*

Part 2: *The structure of a TransBASIC module – each TransBASIC module follows a format designed to make them simple to create and "mergeable" with other modules.*

Part 3: *ROM routines used by TransBASIC – many modules make use of ROM routines burried inside the Commodore 64. Part 3 explains how to use these routines when creating new modules.*

Part 4: *Using Numeric Expressions – details on how to make use of the evaluate expression ROM routine.*

Part 5: *Assembler Compatibility – TransBASIC modules are written in PAL Assembler format. Techniques for porting them to another assembler were discussed here.*

Part 6: *The USE Command – The command 'ADD' merges TransBASIC modules into text space. However, as more modules are ADDed, merging gets slow. The USE command was written to speed things up. USE also counts the number of modules USEd and updates line 95 automatically.*

Part 7: *The TransBASIC kernel uses all of the 64's system vectors. Should two or more modules attempt to alter any vector, a potential crash situation exists. Part 7 deals with avoiding this situation.*

TransBASIC Part 8

Our first module this issue (Program 1) is called OLD. It was written by Joel M. Rubin of San Francisco, California, and consists of one statement, also called OLD. Like the UNNEW and OLD commands of other utilities, OLD restores a BASIC program that you have lost through inadvertent use of the NEW command. Joel will have more modules appearing in future issues.

The second module is INPN & INPA (Program 2). It provides controlled input of either numeric (INPN) or alphanumeric (INPA) characters. This one is by Wayne Happ of North Babylon, New York (who also wrote the SELECT module, discussed below). Wayne notes that it would be easy to modify the section of code that screens the input characters, if the particular selections made in this published version do not exactly meet your programming requirements. This module also makes use of part of the INLINE module (see below), so if you're typing them in, note that some code is duplicated, and there's no need to enter it twice if you're willing do a little juggling.

SELECT (Program 3) is a short but interesting module that provides a structure not unlike the SWITCH structure of C or the CASE structure of Pascal. SELECT is not a closed, formal structure -- this being BASIC, after all -- but it should be helpful in a lot of instances where an unwieldy series of IF–THEN statements is the only alternative. The SELECT structure begins with SELECT and ends with ENDSELECT, with an arbitrary number of WHEN statements and an optional OTHERWISE statement in–between. However, there is nothing to prevent you using other kinds of statements inside the structure, if you so desire, or even -- since SELECT does not use the stack -- distributing the component statements in various subroutines or FOR–NEXT loops. Formal structures don't allow these kinds of liberties, and so can keep you out of trouble; on the other hand, you might be able to make good use of the freedom SELECT offers you.

MC GRAPHICS (Program 4) is another big module from Darren Spruyt of Gravenhurst, Ontario, whose work has appeared in this column in each of the past two issues. This module gives a battery of twelve commands that make multicolour hi–res graphics easy to use, rather than a programming headache. In multicolour hi–res,

you have the choice of four colours in each low-res pixel (4 by 8 multicolour pixels), rather than the two colours allowed in regular hi-res (though one of the four is the background colour, common to the whole screen). The drawback of multicolour is a loss of resolution - you get a 160 by 200 pixel screen instead of the 320 by 200 pixels of regular hi-res. However, this is still enough to allow good detail.

One problem with programs that run in graphics modes other than the default text mode is that a syntax or other error can leave you with a mess on your screen and no obvious way to get out of it. To get around this, the MC GRAPHICS module reroutes the error vector when the hi-res screen is turned on, and returns the screen to normal before handling the error. The error trap is deactivated when the hi-res is switched off. This will normally introduce no additional problems, but if you want to intercept the error vector for your own purposes, be sure to do so only when the multicolour is off. By the way, if you happen to be writing hi-res modules yourself, whether multicolour or not, it would be a good idea to use the same error-trapping procedure, or even the same routines, as those employed by MC GRAPHICS.

Finally this month we have INLINE (Program 5), a variant of BASIC's INPUT command that does away with the question-mark prompt and also allows all punctuation mark characters to be input. This one works only from the keyboard - maybe in a future issue we'll run a sequel that does the same thing for reading from files.

New Commands

INLINE (Type: Statement Cat #: 030)
Line Range: 3454-3528
Module: INLINE
Example: INLINE "SAY SOMETHING: ";A$
Identical in syntax and operation to the regular INPUT statement except that: the question mark prompt is not given; only one variable may be input and that must be a string variable; commas, colons and semicolons are accepted as input.

OLD (Type: Statement Cat #: 142)
Line Range: 10190-10216
Module: OLD
Example: OLD
This command will restore a BASIC program in memory after a NEW has accidentally been given or the computer has been reset (in the latter case, you will have to first reset TransBASIC with SYS 49155 or POKE 49152,96, then re-enable it with SYS 49152). The command will not function predictably if any syntax errors occur after the NEW.

INPN (Type: Statement Cat #: 143)
Line Range: 10224-10386
Module: INPN & INPA
Example: INPN "YOUR PHONE NUMBER: ";PN$
This is a controlled input command that accepts only numeric characters and some punctuation (space, period, plus, minus). The cursor is a non-flashing underline character. The only allowed control characters are DEL and RETURN. There is no automatic question mark prompt; and the string prompt (as in the example) is optional.

INPA (Type: Statement Cat #: 144)
Line Range: 10218-10386
Module: INPN & INPA
Example: INPA "YOUR NAME: ";PN$
This is a controlled input command that accepts only numeric characters, upper and lower case alphabetics and some punctuation (space, period, plus, minus, and the characters in the ASCII range 58 through 64). The colon (ASCII 58) and semicolon (ASCII 59) are accepted as input like other characters, not as terminators. The cursor is a non-flashing underline character. The only allowed control characters are DEL and RETURN. There is no automatic question mark prompt; and the string prompt (as in the example) is optional. The variable must be of string type.

SELECT (Type: Statement Cat #: 145)
Line Range: 10388-10410
Module: SELECT
Example: SELECT
Example: SELECT A(3)-1
This statement begins the SELECT structure. If no parameter is given (first example), the structure operates in logical mode; when a parameter is present (second example) it operates in comparison mode. For further details see the entry for the WHEN statement (146).

WHEN (Type: Statement Cat #: 146)
Line Range: 10412-10446
Module: SELECT
Example: WHEN A=B PRINT "EQUALITY"
Example: WHEN A PRINT "CASE";A
This statement is part of the SELECT structure (see SELECT, #145). The first example illustrates the syntax when the structure is in logical mode: the WHEN is followed by a logical expression which, as in the IF statement, controls whether the remainder of the program line is executed. The second example illustrates the syntax of the comparison mode: the WHEN is followed by an expression whose result is compared with the value of the expression accompanying the SELECT statement, and the remainder of the line is executed if the two values are equal. If the WHEN test expression is successful (and thus the remainder of the program line is executed), the SELECT structure is disabled: further WHEN statements and OTHERWISE (#147) statements will have no effect.

OTHERWISE (Type: Statement Cat #: 147)
Line Range: 10448-10458
Module: SELECT
Example: OTHERWISE PRINT "NO LUCK"
The OTHERWISE statement is an optional component of the SELECT structure (see SELECT, #145). Statements following on the same line will be executed only if no WHEN statement (#146) has been successful. The OTHERWISE statement disables the SELECT structure: further WHEN statements and OTHERWISE statements will have no effect.

ENDSELECT (Type: Statement Cat #: 148)
Line Range: 10438-10446
Module: SELECT
Example: ENDSELECT
This statement terminates the SELECT structure (see SELECT, #145).

HCLR (Type: Statement Cat #: 149)
Line Range: 11106–11134
Module: MC GRAPHICS
Example: HCLR
This statement clears the high–resolution screen at address $E000–$FFFF by filling that area of memory with zero bytes.

MCON (Type: Statement Cat #: 150)
Line Range: 11068–11104
Module: MC GRAPHICS
Example: MCON
This statement enables the multicolour hi–res screen at address $E000, with video matrix at $D800. Current values of background colour, text colour and character set location are saved, to be restored later with the HOFF (#151) command.

HOFF (Type: Statement Cat #: 151)
Line Range: 11154–11198
Module: MC GRAPHICS
Example: HOFF
This statement disables hi–res and multicolour modes, sets the video matrix (low–res screen) to its usual location of $0400, and restores previous low–res values of background colour, text colour and character set location, as saved with the MCON (#150) command.

MSET (Type: Statement Cat #: 152)
Line Range: 10746–10762
Module: MC GRAPHICS
Example: MSET 100,30,2
This statement sets the specified screen location (in the example x = 100, y = 30) to the specified colour. The colour in the example is number 2, as set in the MCOLOR command (#159).

MTEXT (Type: Statement Cat #: 153)
Line Range: 11784–11996
Module: MC GRAPHICS
Example: MTEXT 10,40,1,2,3,"HELLO"
This command writes the given string, in a specified colour, to a location on the multicolour high–res screen. The text may be magnified by an integer factor in both the X and Y dimensions (no magnification results in a character size of 8 by 8 multicolour pixels). The order of parameters is: x location, y location, x magnification, y magnification, colour and the string to be printed. The character shapes are drawn from the Upper Case/Graphics ROM character set. The example prints the text "HELLO" at coordinates x = 10, y = 40, with no magnification in the X dimension, and double magnification in the Y dimension, using colour 3 as set by the MCOLOR command (#159).

MCIRCLE (Type: Statement Cat #: 154)
Line Range: 11362–11782
Module: MC GRAPHICS
Example: MCIRCLE 80,100,50,1,1,2
This command draws a circle with a specified radius to a location on the multicolour high–res screen, using a specified colour. The X and Y dimensions of the circle may be adjusted (creating an oval) with separate multipliers, whose value should lie between 0 and 1. The order of parameters is: x location, y location, radius, x magnification, y magnification and colour. The example draws a circle of radius 50 at coordinates x = 80, y = 100, with no magnification in either the x or y dimensions, using colour 2 as set by the MCOLOR command (#159).

MDISC (Type: Statement Cat #: 155)
Line Range: 11356–11782
Module: MC GRAPHICS
Example: MDISC 80,100,60,1,.8,1
This command is identical to MCIRCLE (#154) except that the circle is filled instead of being drawn as an outline.

MRECT (Type: Statement Cat #: 156)
Line Range: 11270–11318
Module: MC GRAPHICS
Example: MRECT 30,40,50,15,1
This statement draws a rectangle specified by the coordinates of two diagonally opposite corners, in a specified colour. It does not matter which corners are specified. The example draws a rectangle whose lower left corner is at x = 30, y = 40, and whose upper right corner is at x = 50, y = 15, using colour 1 as set by the MCOLOR command (#159).

MBOX (Type: Statement Cat #: 157)
Line Range: 11320–11354
Module: MC GRAPHICS
Example: MBOX X1,Y1,X2,Y2,C
This command is identical to MRECT (#156) except that the rectangle is filled instead of being drawn as an outline.

MDRAW (Type: Statement Cat #: 158)
Line Range: 10472–10626
Module: MC GRAPHICS
Example: MDRAW 13,111,77,99,0
This command draws a line between two points in a specified colour. The example draws a line from x = 13, y = 111 to x = 77, y = 99, in the background colour as set by the MCOLOR command (#159).

MCOLOR (Type: Statement Cat #: 159)
Line Range: 10954–11066
Module: MC GRAPHICS
Example: MCOLOR 10,10,25,20,0,2,3,6
This command sets the four colours available in the multicolour palette for a specified screen region as specified by the low–res co–ordinates (x = 0 to 39, y = 0 to 24) of its upper left and lower right corners. The example sets colour 0 (the background colour) to black (0), colour 1 to red (2), colour 2 to cyan (3), and colour 3 to blue (6), for the region whose upper left corner is at x = 10, y = 10, and whose lower right corner is at x = 25, y = 20. Colour 0, the background colour, is different from the others in that it is set for the whole screen at once, and not only for the specified region.

MCHK((Type: Function Cat #: 160)
Line Range: 11136–11152
Module: MC GRAPHICS
Example: PRINT MCHK(100,yc)
This function returns the colour (as set by the MCOLOR command, #159) of the specified point (in the example, x = 100, y = yc).

Modules So Far

TransBASIC Modules that have appeared so far (Instalments 1 to 7)

TransBASIC #1

TB/KERNEL

Statements: 2 Functions: 0 Keyword Characters: 8

000 S/IF	Modified IF to work with TransBASIC
001 S/ELSE	Part of IF-ELSE construct
002 S/EXIT	Disable current TransBASIC dialect

SCREEN THINGS

Statements: 5 Functions: 0 Keyword Characters: 22

013 S/GROUND	Set background colour
014 S/FRAME	Set border colour
015 S/TEXT	Set text colour
016 S/CRAM	Fill colour memory with value
017 S/CLS	Clear screen, or screen line range

TransBASIC #2

DOKE & DEEK

Statements: 1 Functions: 1 Keyword Characters: 9

007 S/DOKE	Poke a 16-bit value
008 F/DEEK(	Peek a 16-bit value

BIT TWIDDLERS

Statements: 3 Functions: 0 Keyword Characters: 12

009 S/SET	Set specified bit at address
010 S/CLEAR	Clear specified bit at address
011 S/FLIP	Flip specified bit at address

CHECK & AWAIT

Statements: 0 Functions: 2 Keyword Characters: 12

018 F/CHECK(	Check keyboard for valid character
019 F/AWAIT(	Wait for valid character from keyboard

KEYWORDS

Statements: 1 Functions: 0 Keyword Characters: 8

059 S/KEYWORDS Print currently active TransBASIC keywords

TransBASIC #3

CURSOR POSITION

Statements: 1 Functions: 1 Keyword Characters: 10

004 S/CURSOR	Move cursor to specified row and column
005 F/CLOC	Return cursor location

SET SPRITES

Statements: 6 Functions: 0 Keyword Characters: 27

031 S/COLSPR	Set colour of sprite
032 S/SSPR	Turn on a sprite
033 S/CSPR	Turn off a sprite
034 S/XSPR	Move sprite to specified x-position
035 S/YSPR	Move sprite to specified y-position
036 S/XYSPR	Move sprite to specified xy-position

WITHIN

Statements: 0 Functions: 1 Keyword Characters: 7

040 F/WITHIN(Return true if value lies within specified range

READ SPRITES

Statements: 0 Functions: 2 Keyword Characters: 10

041 F/XLOC(	Return x-position of specified sprite
042 F/YLOC(	Return y-position of specified sprite

TransBASIC #4

STRIP & CLEAN

Statements: 0 Functions: 2 Keyword Characters: 14

045 F/STRIP$(	Remove non-alphanumerics from string
046 F/CLEAN$(	Remove non-blank non-alphanumerics from string

SCROLLS

Statements: 4 Functions: 0 Keyword Characters: 24

067 S/USCROL	Scroll screen area up one row
068 S/DSCROL	Scroll screen area down one row
069 S/LSCROL	Scroll screen area left one row
070 S/RSCROL	Scroll screen area right one row

TransBASIC #5

LABELS

Statements: 5 Functions: 0 Keyword Characters: 24

073 S/L.	Label a line
074 S/LGOTO	GOTO a labelled line
075 S/LGOSUB	GOSUB to a labelled line
076 S/SGOTO	GOTO a line whose label matches a string
077 S/SGOSUB	GOSUB to a line whose label matches a string

TOKEN & VAR

Statements: 0 Functions: 2 Keyword Characters: 11

078 F/TOKEN$(Return tokenized version of argument string
079 F/VAR(Return address of data of named variable

INSTRING

Statements: 0 Functions: 1 Keyword Characters: 6

080 F/INSTR(Search string 1 for string 2. Boolean options

PLACE

Statements: 0 Functions: 1 Keyword Characters: 6

081 F/PLACE(Search string 1 for string 2 from specified position

ARCFUNCTIONS

Statements: 0 Functions: 2 Keyword Characters: 8

082 F/ASN(Return arcsine of argument
083 F/ACS(Return arccosine of argument

PRINTAT

Statements: 1 Functions: 0 Keyword Characters: 6

084 S/PRINT@ Print at specified cursor position

SOUND THINGS

Statements: 28 Functions: 4 Keyword Characters: 126

085 S/CLESID Clear SID chip
086 S/FREQ Set SID voice frequency
087 S/PUWID Set pulse width
088 S/FIFREQ Set filter cutoff frequency
089 S/ADPUL Add pulse to waveform
090 S/ADSAW Add sawtooth to waveform
091 S/ADTRI Add triangle to waveform
092 S/NOWAV Clear waveform register
093 S/NOI Set noise waveform
094 S/PUL Set pulse waveform
095 S/SAW Set sawtooth waveform
096 S/TRI Set triangle waveform
097 S/TEST Set/clear waveform register test bit
098 S/RING Set/clear ring modulation
099 S/SYNC Set/clear synchronization
100 S/GATE Set/clear gate bit
101 S/ATT Set attack
102 S/DEC Set decay
103 S/SUS Set sustain
104 S/REL Set release
105 S/RESON Set filter resonance
106 S/VOL Set volume
107 S/FILT Set/clear filter
108 S/TRDOFF Turn off oscillator 3
109 S/TRDON Turn on oscillator 3
110 S/HP Turn high-pass filter on/off
111 S/BP Turn band-pass filter on/off
112 S/LP Turn low-pass filter on/off
113 POTX Return value of port 1 game paddle
114 POTY Return value of port 2 game paddle
115 OSC3 Return value of oscillator 3 output
116 ENV3 Return value of oscillator 3 envelope generator

TransBASIC #6

USE

Statements: 1 Functions: 0 Keyword Characters: 3

117 S/USE Fast-merge programs, TransBASIC modules

MOVE & FILL

Statements: 2 Functions: 0 Keyword Characters: 8

118 S/MOVE Move area of memory
119 S/FILL Fill area of memory with specified value

DOS SUPPORT

Statements: 5 Functions: 2 Keyword Characters: 24

123 S/CAT List directory to current output device
124 S/DOS Send a command to disk
125 S/DEV Set default disk device number
126 S/DLOAD Load program from default drive
127 S/DSAVE Save program to default drive
128 F/DS$ Return disk error string
129 F/DS Return disk error number

LINE CALC

Statements: 2 Functions: 1 Keyword Characters: 13

130 S/JUMP Goto program line at specified address
131 S/CALL Call subroutine at specified address
132 F/LINE(Determine address of specified line

BEEP

Statements: 1 Functions: 0 Keyword Characters: 4

133 S/BEEP Produce a beep tone

TransBASIC #7

RANDOM

Statements: 0 Functions: 1 Keyword Characters: 7

027 F/RANDOM(Return random integer within range

PHRASE SPLITTERS

Statements: 0 Functions: 2 Keyword Characters: 11

028 F/FIRST$(Return first word of string
029 F/BF$ Return all but first word of string

PRG MANAGEMENT

Statements: 3 Functions: 0 Keyword Characters: 10

136 S/AUTO Generate line numbers automatically
137 S/DEL Delete program line(s)
138 S/REN Renumber BASIC program

COMPUTED CMDS

Statements: 3 Functions: 0 Keyword Characters: 16

139 S/RESTORE Restore DATA pointer to computed line number
140 S/GOSUB Call subroutine at computed line number
141 S/GOTO Transfer execution to computed line number

Program 1: OLD

```
IJ  0 rem old (j. rubin, sept/85)      :
FH  1 :
AI  2 rem  1 statement, 0 functions
HH  3 :
DO  4 rem keyword characters: 3
JH  5 :
NJ  6 rem keyword     routine     line      ser #
JP  7 rem s/old       ol          10190     142
MH  8 :
CO  9 rem================================
OH  10 :
OH  141 .asc "olD"
FG  1441 .word ol-1
CD  10190 ol      ldy  #1        ;make first line hi
DO  10192         tya            ; link non-0
DC  10194         sta  (43),y
JB  10196         jsr  $a533     ;relink program
JK  10198         clc            ;($22) points 1
JO  10200         lda  $22       ; byte past end
OC  10202         adc  #2        ; of program. add
AM  10204         sta  45        ; 2, and write to
DF  10206         lda  $23       ; start of vars
NL  10208         adc  #0
IC  10210         sta  46
DF  10212         jsr  $a660     ;clr
NG  10214         jmp  $e386     ;ready
OF  10216 ;
```

Program 2: INPA and INPN

```
OP  0 rem inpa & inpn (w.happ, 1985)    :
FH  1 :
DH  2 rem 2 statements, 0 functions
HH  3 :
IO  4 rem keyword characters: 8
JH  5 :
NJ  6 rem keyword     routine     line      ser #
BG  7 rem s/inpa      npa         10218     143
DL  8 rem s/inpn      npn         10224     144
NH  9 :
PO  10 rem================================
PH  11 :
MM  142 .asc "inpAinpN"
BB  1442 .word npa-1,npn-1
BO  3478 inl2     jsr  $ad8f
FH  3480          sta  $49
PO  3482          sty  $4a
JF  3484          lda  $7a
OL  3486          ldy  $7b
IJ  3488          sta  $4b
NP  3490          sty  $4c
NI  3492          ldx  $43
GJ  3494          ldy  $44
PP  3496          stx  $7a
IA  3498          sty  $7b
MK  3500          jsr  $73
JJ  3502          lda  #0
PH  3504          sta  $07
EI  3506          sta  $08
BH  3508          lda  $7a
GN  3510          ldy  $7b
FP  3512          jsr  $b48d
CC  3514          jsr  $b7e2
DD  3516          jsr  $a9da
IH  3518          lda  $4b
NN  3520          ldy  $4c
NL  3522          sta  $7a
CC  3524          sty  $7b
CL  3526          rts
OD  3528 ;
GK  10218 npa     lda  #"["      ;last valid char + 1
AC  10220         .byte $2c      ;'bit'
EG  10222 ;
OJ  10224 npn     lda  #":"      ;last valid char + 1
AF  10226         sta  t2
CP  10228         jsr  $79       ;test arg present
BM  10230         bne  np1       ; yes
FP  10232         jmp  $af08     ;'syntax'
AB  10234 np1     cmp #$22       ;test for quote
KN  10236         bne  np2       ; no
KJ  10238         jsr  $aebd     ;set up prompt str
NO  10240         lda  #";"      ;check semicolon
KK  10242         jsr  $aeff
EB  10244         jsr  $ab21     ;print prompt str
EP  10246 np2     jsr  $b3a6     ;check prg mode
GO  10248         lda  #","      ;put comma before
LE  10250         sta  $01ff     ; input buffer
IP  10252         ldx  #0        ;init char counter
KM  10254         stx  t3
BM  10256         stx  $11       ;set input flag
BM  10258 np3     lda  #$a4      ;print underline
CI  10260         jsr  $ffd2     ; (cursor)
HE  10262 np4     jsr  $ffe4     ;get character
OJ  10264         cmp #$0d       ;test cr
PA  10266         beq  np8       ; yes
FK  10268         cmp #$14       ;test delete
AB  10270         beq  np7       ; yes
HE  10272         cmp #$20       ;test space
BB  10274         beq  np6       ; yes
GA  10276         cmp #"."       ;test period
FB  10278         beq  np6       ; yes
CM  10280         cmp #"+"       ;test plus
JB  10282         beq  np6       ; yes
JE  10284         cmp #"-"       ;test minus
NB  10286         beq  np6       ; yes
JG  10288         ldx  t2        ;test inpa
IA  10290         cpx  #"["
LB  10292         bne  np5       ; no
EN  10294         cmp #"A"       ;test upper case
GP  10296         bcc  np5       ; no
GC  10298         cmp #$db
NO  10300         bcc  np6       ;yes
HA  10302 np5     cmp #"0"       ;test if in range
LP  10304         bcc  np4       ; no
HK  10306         cmp t2
PD  10308         bcs  np4       ; no
AF  10310 np6     ldx  $d3       ;get cursor column
OH  10312         cpx  #$4f      ;test < 79
FE  10314         bcs  np4       ; no
HF  10316         pha            ;print cursor left
MN  10318         lda  #$9d
JO  10320         jsr  $ffd2
KN  10322         pla
AC  10324         jsr  $ffd2     ;print input char
BL  10326         ldx  t3        ;get buffer index
```

```
HD 10328       sta  $0200,x  ;char to buffer
NF 10330       inc  t3       ;bump index
DE 10332       bne  np3      ;get another char
DI 10334 np7   ldx  t3       ;test index > 0
JF 10336       beq  np4      ; no
II 10338       jsr  $ffd2    ;delete cursor
GN 10340       jsr  $ffd2    ;delete character
JA 10342       dec  t3       ;back up index
JK 10344       bpl  np3      ;handle next input
AG 10346 np8   lda  #$14     ;delete cursor
FA 10348       jsr  $ffd2
JM 10350       ldx  t3       ;get buffer index
KJ 10352       jsr  $aaca    ;print cr
FD 10354       stx  $43      ;$1ff to input ptr
CK 10356       sty  $44
MP 10358       ldx  t3       ;test input null
LG 10360       bne  np9      ; no
AG 10362       jmp  $abf3    ;no assignment
PI 10364 np9   jsr  $b08b    ;find variable
KP 10366       pha
JH 10368       jsr  $79      ;test end of line
CG 10370       beq  np10     ;yes
BI 10372       jmp  $af08    ;'syntax'
OI 10374 np10  pla
IL 10376       ldx  t2       ;test alpha
AG 10378       cpx  #"["
HB 10380       bne  np11     ; no
FJ 10382       jmp  inl2     ;enter inline rtn
IE 10384 np11  jmp  $ac18    ;enter rom input
IA 10386 ;
```

```
PH 10412 whn   beq  oth1     ;error - no arg
DI 10414       jsr  $ad8a    ;eval expression
EC 10416       lda  sflg1    ;test sel on
HK 10418       beq  wn1      ; no
OK 10420       lda  sflg2    ;test for test expr
JI 10422       bne  wn2      ; yes
CD 10424       lda  $61      ;test 'when' expr
PL 10426       bne  wn3      ; true
OJ 10428 wn1   jmp  $a93b    ;enter rem routine
OG 10430 wn2   lda  #<seltst ;compare test
OC 10432       ldy  #>seltst ; expression
MB 10434       jsr  $bc5b    ; with fac 1
JL 10436       bne  wn1      ;not equal, ignore
FK 10438 wn3   lda  #0       ;flag - select off
OH 10440       sta  sflg1
HK 10442       jsr  $79      ;execute statement
IN 10444       jmp  $a940
EE 10446 ;
PJ 10448 oth   beq  oth1     ;error - no arg
GE 10450       lda  sflg1    ;test sel on
KK 10452       bne  wn3      ; yes
LM 10454       beq  wn1      ; no
KN 10456 oth1  jmp  $af08    ;syntax
AF 10458 ;
PE 10460 sflg1 .byte 0
EF 10462 ;
FF 10464 sflg2 .byte 0
IF 10466 ;
LA 10468 seltst *=*+5
MF 10470 ;
```

Program 3: SELECT

```
JC 0 rem select (wayne happ, 1985)    :
FH 1 :
JH 2 rem 4 statements, 0 functions
HH 3 :
CF 4 rem keyword characters: 28
JH 5 :
NJ 6 rem keyword       routine   line     ser #
BP 7 rem s/select      sel       10388    145
MI 8 rem s/when        whn       10412    146
EP 9 rem s/otherwise   oth       10448    147
FF 10 rem s/endselect  wn3       10438    148
PH 11 :
KD 12 rem ================================
BI 13 :
BP 143 .asc "selecTwheN"
PD 144 .asc "otherwisEendselecT"
HB 1143 .word sel-1,whn-1
NP 1144 .word oth-1,wn3-1
JL 10388 sel   bne  sel1     ;set sflg2 if
FE 10390       lda  #0       ; argument present
FC 10392 sel1  sta  sflg2
GO 10394       beq  sel2     ;skip if no arg
OL 10396       jsr  $ad8a    ;eval expr
HD 10398       ldx  #<seltst ;store result
EA 10400       ldy  #>seltst ; to seltst
JC 10402       jsr  $bbd4
DP 10404 sel2  lda  #1       ;flag - select on
MF 10406       sta  sflg1
EJ 10408       rts
AC 10410 ;
```

Program 4: MC GRAPHICS

Darren's done it again! This next TB module comes to 85 Blocks of source code on Transactor Disk #11! It's been printed here for reference, but we didn't include the Verifizer codes – we couldn't imagine anyone actually typing it in, except Darren of course. M.Ed.

```
0 rem mc graphics (d. spruyt, 1985)  :
1 :
2 rem 11 statements,  1 function
3 :
4 rem keyword characters: 57
5 :
6 rem keyword      routine   line     ser #
7 rem s/mcon       ho        11068    149
8 rem s/mset       mcset     10746    150
9 rem s/hoff       mcrstr    11154    151
10 rem s/mdraw     mcdra     10472    152
11 rem s/mrect     mcrec     11270    153
12 rem s/mcircle   mccir     11362    154
13 rem s/mtext     mctxt     11784    155
14 rem s/hclr      hcl       11106    156
15 rem s/mcolor    mccol     10954    157
16 rem s/mbox      mcbox     11320    158
17 rem s/mdisc     mcdis     11356    159
18 rem f/mchk(     mchk      11136    160
19 :
20 rem ================================
21 :
145 .asc "mcoNmseThofF"
146 .asc "mdraWmrecTmcirclE"
147 .asc "mtexThclRmcoloR"
148 .asc "mboXmdisC"
622 .asc "mchk":.byte $80+"("
1145 .word ho-1,mcset-1,mcrstr-1
1146 .word mcdra-1,mcrec-1,mccir-1
1147 .word mctxt-1,hcl-1,mccol-1
1148 .word mcbox-1,mcdis-1
1622 .word mchk-1
```

```
10472 mcdra  lda  #0       ;select non-
10474        sta  t3       ; duplicating
10476        jsr  mcget5   ;get 5 params
10478 mdr1   lda  #0       ;zero neg/pos
10480        sta  $ab      ;dir x flag
10482        sta  $ae      ;dir y flag
10484        lda  t5       ;get y2 value
10486        sec           ;subtract y1
10488        sbc  $a6      ; value and skip
10490        bcs  mdr2     ; over the twos
10492        eor  #$ff     ; complement
10494        adc  #1       ; if y2>y1
10496        dec  $ae      ;set neg flag
10498 mdr2   sta  $ad      ;save y diff
10500        lda  $21      ;get x2 value
10502        sec           ;subtract x1
10504        sbc  $a8      ; value and skip
10506        bcs  mdr3     ; over the 2's
10508        eor  #$ff     ; complementing
10510        adc  #1       ; and negative
10512        dec  $ab
10514 mdr3   sta  $aa      ;save x diff
10516        lda  $ad      ;get y diff
10518        cmp  $aa      ;compare x diff
10520        bcs  mdr4     ; y bigger
10522        lda  $aa      ; x bigger
10524 mdr4   sta  $af      ;save biggest
10526        lda  #0       ;zero location
10528        sta  $a3
10530        lda  $aa      ;get x diff
10532        sta  $a4      ;save
10534        jsr  mcdiv    ;divide $a4 val
```

```
10536          lda  $a3        ; by $af val
10538          sta  $a9        ;copy 16-bit
10540          lda  $a4        ; result to ($a9)
10542          sta  $aa
10544          lda  #0         ;zero location
10546          sta  $a3
10548          lda  $ad        ;perform same
10550          sta  $a4        ; division for y
10552          jsr  mcdiv      ; with result in
10554          lda  $a3        ; ($ac)
10556          sta  $ac
10558          lda  $a4
10560          sta  $ad
10562          lda  #$80       ;set plot values
10564          sta  $a7        ;lower fractions
10566          sta  $a5        ; to 0.5
10568          ldx  #$a9       ;complement ($a9)
10570          jsr  mcneg      ; if needed
10572          ldx  #$ac       ;complement ($ac)
10574          jsr  mcneg      ; if needed
10576          ldx  $af        ;number of steps
10578 mdr5     lda  $a8        ;copy x value
10580          sta  $b5        ; to plot loc'n
10582          lda  $a6        ;copy y value
10584          sta  $b4        ; to plot loc'n
10586          lda  $b0        ;colour-pattern
10588          jsr  mcplo      ;plot point
10590          lda  $a7        ;accumulate
10592          clc             ; fractions to
10594          adc  $a9        ; x and y values
10596          sta  $a7
10598          lda  $a8
10600          adc  $aa
10602          sta  $a8
10604          lda  $a5
10606          clc
10608          adc  $ac
10610          sta  $a5
10612          lda  $a6
10614          adc  $ad
10616          sta  $a6
10618          dex             ;dec plots-to-do
10620          cpx  #$ff       ; count and loop
10622          bne  mdr5       ; if necessary.
10624          rts
10626 ;
10628 mcplo    sta  $a4        ;colour-pattern
10630          sta  t4
10632          lda  t3         ;get mode
10634          beq  mcp2       ; normal draw
10636          bmi  mcp1       ; x parallel
10638          jsr  mcp2       ; y parallel
10640          lda  $b4        ;get y value
10642          clc
10644          adc  t6         ;add parallel
10646          sta  $b4        ; value
10648          lda  t4         ;get colour
10650          sta  $a4
10652          jmp  mcp2       ;replot parallel
10654 mcp1     jsr  mcp2       ;plot initial
10656          lda  $b5        ;get x value
10658          clc
10660          adc  t6         ;add parallel
10662          sta  $b5        ; value
10664          lda  t4         ;get colour
10666          sta  $a4        ; replot parallel
10668 mcp2     lda  $b5        ;get x value
10670          cmp  #$a0       ;test off screen
10672          bcc  mcp4       ; no
10674 mcp3     rts             ;â
10676 mcp4     lda  $b4        ;get y value
10678          cmp  #$c8       ;test off screen
10680          bcs  mcp3       ;yes
10682          tya             ;push .y
10684          pha
10686          jsr  ptr19      ;set up pointers
10688          lda  $b5        ;get low bits of
10690          and  #3         ; x value to .y
10692          tay
10694          lda  mctbl2,y   ;get mask patt
10696          pha             ;push it
10698          lda  $b5        ;copy low bit
10700          lsr             ; into sr as
10702          php             ; carry and save
10704          lsr             ;next low bit
10706          lda  $a4        ;colour pattern
10708          bcs  mcp5       ; 2/3 pattern
10710          asl             ; 0/1 pattern
10712          asl             ;lo nybble to hi
10714          asl
10716          asl
10718 mcp5     plp             ;retrieve carry
10720          bcs  mcp6       ;skip if 3/1 pat
10722          asl             ;shift bits 2
10724          asl             ; more places up
10726 mcp6     sta  $a4        ;save value
10728          pla             ;pull mask value
10730          ldy  #0         ;init index
10732          jsr  ropen      ;open ram
10734          and  ($19),y    ;and value with
10736          ora  $a4        ; mask, or patt
10738          sta  ($19),y    ;put value back
10740          jsr  rclose     ;close ram
10742          jmp  mcr5       ;finish up
10744 ;
10746 mcset    lda  #0         ;select non-
10748          sta  t3         ; duplicating
10750          jsr  mcg1       ;get x value,
10752          sta  $b5
10754          jsr  mcget      ; y value,
10756          sta  $b4
10758          jsr  mcget      ; colour pattern
10760          jmp  mcplo      ;plot point
10762 ;
10764 mcget    jsr  $aefd      ;check comma
10766 mcg1     jsr  $b79e      ;eval 1 byte
10768          txa             ;parameter to .x
10770          rts
10772 ;
10774 mcdiv    ldx  #$10       ;divide $a4 val
10776          lda  #0         ; by $af val with
10778          asl  $a4        ; shift and
10780 mcdv1    rol  ; subtract.
10782          bcs  mcdv3      ; hi byte of
10784          cmp  $af        ; result will be
10786          bcc  mcdv2      ; 0 or 1
10788          sbc  $af
10790 mcdv2    rol  $a3
10792          rol  $a4
10794          dex
10796          bne  mcdv1
10798          rts
10800 mcdv3    sbc  $af
10802          sec
10804          bcs  mcdv2
10806 ;
10808 mcneg    lda  2,x        ;perform twos
10810          bpl  mcn1       ; complement of
10812          sec             ; count value if
10814          lda  0,x        ; count direction
10816          eor  #$ff       ; is negative
10818          adc  #0
10820          sta  0,x
10822          lda  1,x
10824          eor  #$ff
10826          adc  #0
10828          sta  1,x
10830 mcn1     rts
10832 ;
10834 ptr19    lda  $b4        ;divide y value
10836          lsr             ; by 4, clear lsb
10838          lsr
10840          and  #$fe
10842          tay             ;up values for
10844          lda  mctbl1,y   ; hi-res map from
10846          sta  $19        ; table, make ptr
10848          lda  mctbl1+1,y in ($19)
10850          sta  $1a
10852          lda  $b4        ;save bits 0-2 of
10854          and  #7         ; y val (relative
10856          sta  $a3        ; scan line 0-7)
10858          lda  $b5        ;6 high bits of
10860          and  #$fc       ; x value times 2
10862          asl             ; (screen column)
10864          bcc  p19r1
10866          inc  $1a
10868 p19r1    clc             ;adjust pointer
10870          ora  $a3        ; to true pos'n
10872          adc  $19        ; in 8x8 block
10874          sta  $19
10876          bcc  p19r2
10878          inc  $1a
10880 p19r2    rts
10882 ;
10884 mcrea    lda  $b4        ;test y value
10886          cmp  #$c8       ; in range
10888          bcc  mcr2       ; yes
10890 mcr1     lda  #$ff       ;set error flag
10892          rts
10894 mcr2     lda  $b5        ;test x value
10896          cmp  #$a0       ; in range
10898          bcs  mcr1       ; no
10900          tya             ;push .y
10902          pha
10904          jsr  ptr19      ;create pointer
10906          ldy  #0         ;init index
10908          lda  $b5        ;get and save
10910          lsr             ;save lsb to sr
10912          php             ;push sr
10914          lsr             ;next lsb to sr
10916          jsr  ropen      ;open ram
10918          lda  ($19),y    ;get value
10920          jsr  rclose     ;close ram
10922          bcs  mcr3       ;colour 2/3
10924          lsr             ;shift hi nybble
10926          lsr             ; to lo nybble
10928          lsr             ; (colour 0/1)
10930          lsr
10932 mcr3     plp             ;pull x-value lsb
10934          bcs  mcr4       ;colour 1 or 3
10936          lsr             ;shift 2 bits rt.
10938          lsr
10940 mcr4     and  #3         ;save 2 low bits
10942 mcr5     sta  $a3
10944          pla             ;pull .y
10946          tay
10948          lda  $a3        ;retrieve .a
10950 mcr6     rts
10952 ;
10954 mccol    jsr  mcg1       ;set colours
10956          sta  $a7        ;left x value
10958          jsr  mcget      ;
10960          sta  $a8        ;top y value
10962          jsr  mcget      ;right x value
10964          sec             ;calc x diff
10966          sbc  $a7
10968          bcc  mcr6       ;right<left
10970          sta  $a6
10972          jsr  mcget      ;bottom y value
10974          sec             ;calc y diff
10976          sbc  $a8
10978          bcc  mcr6       ;bottom<top
10980          sta  $a5
10982          jsr  mcget      ;set background
10984          sta  $d021
10986          jsr  mcget      ;shift %01 value
10988          asl             ; to hi nybble
10990          asl
10992          asl
10994          asl
10996          sta  $a9        ;temp save
10998          jsr  mcget      ;or with %10 val
11000          and  #$0f
11002          ora  $a9
11004          sta  $a9        ;save
11006          jsr  mcget      ;%11 value
11008          sta  $aa
11010          lda  #0         ;create pointer
11012          sta  $19        ; to colour ram &
11014          lda  #$d8       ; underlying vm
11016          sta  $1a
11018          ldy  $a8        ;adjust pointer
11020 mcc1     dey             ; for position
11022          bmi  mcc2       ; of area to be
11024          jsr  add19      ; filled
11026          jmp  mcc1
11028 mcc2     lda  $19
11030          clc
11032          adc  $a7
11034          sta  $19
11036          bcc  mcc3
11038          inc  $1a        ;o-flow if ness.
11040 mcc3     ldy  $a6        ;write colour to
11042 mcc4     lda  $aa        ; colour ram
11044          sta  ($19),y
11046          jsr  ropen      ;open ram
11048          lda  $a9        ;write colour to
11050          sta  ($19),y    ; vm
11052          jsr  rclose     ;close ram
11054          dey             ;test row done
11056          bpl  mcc4       ; no
11058          jsr  add19      ;advance next row
11060          dec  $a5        ;test all done
11062          bpl  mcc3       ; no
11064          rts
11066 ;
11068 ho       lda  $d011      ;hi-res on
11070          ora  #$20
11072          sta  $d011
11074          lda  $d016      ;multicolour on
11076          ora  #$10
11078          sta  $d016
11080          lda  #$68       ;vm offset $1800,
11082          sta  $d018      ; hires $2000
11084          lda  $dd00      ;select topmost
11086          and  #$fc       ; 16k ram slice
11088          sta  $dd00
11090          lda  $d021      ;save lo-res
11092          sta  mcuvid     ; background,
11094          lda  $0286      ; text colour,
11096          sta  mcuvid+1   ; char and screen
11098          lda  $d018
11100          sta  mcuvid+2
11102          jmp  mcfxer     ;set up err rtn
11104 ;
11106 hcl      lda  #0         ;create pointer
11108          sta  $19        ; to hi-res scrn
11110          lda  #$e0
11112          sta  $1a
11114          lda  #0         ;init .a and
11116          tay             ; .y index
11118          ldx  #$20       ;.x counts pages
11120 hcl1     sta  ($19),y    ;clear 32 pages
11122          iny
11124          bne  hcl1
11126          inc  $1a
11128          dex
11130          bne  hcl1
11132          rts
11134 ;
11136 mchk     jsr  mcg1       ;get x value
11138          sta  $b5
11140          jsr  mcget      ;get y value
11142          sta  $b4
11144          jsr  $aef7      ;check ')'
11146          jsr  mcrea      ;read point
11148          tay             ;conv to fp
11150          jmp  $b3a2
11152 ;
11154 mcrstr    sei
11156          lda  evtmp      ;restore previous
11158          ldy  evtmp+1    ; error vector
11160          sta  $300
11162          sty  $301
11164          cli
11166          lda  $d011      ;hi-res off
11168          and  #$df
11170          sta  $d011
11172          lda  $d016      ;multicolour off
11174          and  #$ef
11176          sta  $d016
11178          lda  $dd00      ;select first 16k
11180          ora  #3         ; ram slice
11182          sta  $dd00
11184          lda  mcuvid     ;restore lo-res
11186          sta  $d021      ; background,
11188          lda  mcuvid+1   ; text colour,
11190          sta  $0286      ; char and screen
11192          lda  mcuvid+2
11194          sta  $d018
11196          rts
11198 ;
11200 mcget5   jsr  mcg1       ;get 5 parameters
11202          sta  $1d        ;x1
11204          sta  $a8
11206          jsr  mcget
11208          sta  $1e        ;y1
11210          sta  $a6
11212          jsr  mcget
11214          sta  $1f        ;x2
11216          sta  $21
11218          jsr  mcget
11220          sta  $20        ;y2
11222          sta  t5
```

```
11224           jsr   mcget
11226           sta   $b0        ;colour
11228           rts
11230 ;
11232 mcfix5    jsr   mcget5
11234           lda   $1f
11236           cmp   $1d
11238           bcs   mcf1
11240           ldx   $1d
11242           sta   $1d
11244           sta   $a8
11246           stx   $1f
11248           stx   $21
11250 mcf1      lda   $20
11252           cmp   $1e
11254           bcs   mcf2
11256           ldx   $1e
11258           sta   $1e
11260           sta   $a6
11262           stx   $20
11264           stx   $t5
11266 mcf2      rts
11268 ;
11270 mcrec     jsr   mcfix5      ;get 5 params
11272           lda   #$f1        ;duplicating on,
11274           sta   t3          ; vertical lines.
11276           lda   $1d         ;save x1
11278           sta   $21
11280           lda   $1f
11282           sec               ;x1 minus x2
11284           sbc   $1d
11286           sta   t6          ;save difference
11288           jsr   mdr1        ;draw line
11290           lda   #1          ;draw duplicate
11292           sta   t3          ; horiz. lines
11294           lda   $1e         ;save y1
11296           sta   $a6
11298           sta   t5
11300           lda   $20         ;y1 minus y2
11302           sec               ;y1 minus y2
11304           sbc   $1e
11306           sta   t6          ;save difference
11308           lda   $1d         ;set x values
11310           sta   $a8         ;draw lines
11312           lda   $1f
11314           sta   $21
11316           jmp   mdr1        ;draw lines
11318 ;
11320 mcbox     jsr   mcfix5      ;get 5 params
11322           lda   #0          ;no duplicating
11324           sta   t3
11326 mcb1      lda   $1e         ;store y1
11328           sta   $a6
11330           sta   t5
11332           lda   $1d         ;store x1
11334           sta   $21
11336           lda   $1f         ;store x2
11338           sta   $a8
11340           jsr   mdr1        ;draw line
11342           inc   $1e         ;bump y value
11344           ldy   $1e         ;test done (=y2)
11346           cpy   $20
11348           beq   mcb1        ; yes
11350           bcc   mcb1        ; no
11352           rts
11354 ;
11356 mcdis     lda   #1          ;flag - disc
11358           .byte $2c
11360 ;
11362 mccir     lda   #0          ;flag - circle
11364           sta   9
11366           ldy   #0          ;no duplicating
11368           sty   t3
11370           jsr   mcg1        ;get x focal pt.
11372           stx   $a5
11374           jsr   mcget       ;get y focal pt.
11376           stx   $a6
11378           jsr   mcget       ;get radius
11380           stx   $a8
11382           jsr   $aefd       ;check comma
11384           jsr   $ad9e       ;eval and save
11386           ldx   #<mymult    ; x factor
11388           ldy   #>mymult
11390           jsr   $bbd7
11392           jsr   $aefd       ;check comma.
11394           jsr   $ad9e       ;eval and save
11396           ldx   #<mxmult    ; y factor
11398           ldy   #>mxmult
11400           jsr   $bbd7
11402           jsr   mcget       ;get colour
11404           stx   $b0
11406           lda   #0          ;clear work area
11408           sta   $a7         ;y position
11410           sta   $ad         ;base lo
11412           sta   $ae         ;base hi
11414 mcd1      ldx   $ae         ;calc base + 2*y + 1
11416           lda   $a7         ; in .a/.x
11418           asl
11420           bcc   mcd2
11422           inx
11424 mcd2      sec
11426           adc   $ad
11428           bcc   mcd3
11430           inx
11432 mcd3      sta   $ab         ;save new y value
11434           stx   $ac
11436           sec               ;calc y-2*x + 1
11438           sbc   $a8         ; in .a/.x
11440           bcs   mcd4
11442           dex
11444           clc
11446 mcd4      sbc   $a8
11448           bcs   mcd5
11450           dex
11452 mcd5      sta   $a9         ;save new x value
11454           stx   $aa
11456           jsr   mcmul       ;adj for factors
11458           lda   9           ;test circle
11460           bne   mcd11       ; no - disc
11462           ldy   #0          ;clear counter
11464           sty   t6          ; for plots
11466 mcd6      lda   $a5         ;get x focal pt.
11468           ldx   mxtype,y    ;push type at pt.
11470           php
11472           ldx   mxadds,y    ;get zp offset
11474           plp               ;test type
11476           beq   mcd7        ; addition
11478           sec               ;subtract value
11480           sbc   0,x         ; at 0,x
11482           jmp   mcd8        ;skip addition
11484 mcd7      clc               ;add value at 0,x
11486           adc   0,x
11488 mcd8      sta   $b5         ;update focal pt.
11490           lda   $a6         ;get y focal pt.
11492           ldx   mytype,y    ;push type at pt.
11494           php
11496           ldx   myadds,y    ;get zp offset
11498           plp               ;test type
11500           beq   mcd9        ; addition
11502           sec               ;subtract value
11504           sbc   0,x         ; at 0,x
11506           jmp   mcd10       ;skip addition
11508 mcd9      clc               ;add value at 0,x
11510           adc   0,x         ;referenced
11512 mcd10     sta   $b4         ;update focal pt.
11514           lda   $b0         ;colour pattern
11516           jsr   mcplo       ;plot point
11518           ldy   t6          ;bump counter
11520           iny
11522           sty   t6
11524           cpy   #8          ;test counter <8
11526           bne   mcd6        ; yes
11528           jmp   mcd17       ;skip disc rtn
11530 mcd11     ldy   #0          ;reset counter
11532           sty   t6
11534           ldy   #$0f        ;set aside data
11536 mcd12     lda   $a3,y       ; to temp storage
11538           sta   mcbuf1,y
11540           dey
11542           bpl   mcd12
11544 mcd13     lda   $a5         ;get x focal pt.
11546           ldy   t6          ;get counter val
11548           clc               ;add offset in
11550           ldx   mctbl3,y    ; 0,x
11552           adc   0,x
11554           sta   $a8         ;save (xleft)
11556           lda   $a5         ;get x focal pt.
11558           sec               ;subtract offset
11560           ldx   mctbl3,y    ; in 0,x
11562           sbc   0,x
11564           sta   $21         ;save (xright)
11566           tya               ;test counter lsb
11568           and   #1
11570           bne   mcd14       ; set
11572           lda   $a6         ;get y focal pt.
11574           clc               ;add offset in
11576           ldx   mctbl4,y    ; 0,x
11578           adc   0,x
11580           jmp   mcd15       ;skip
11582 mcd14     lda   $a6         ;get y focal.
11584           sec               ;subtract offset
11586           ldx   mctbl4,y    ; in 0,x
11588           sbc   0,x
11590 mcd15     sta   $a6         ;save y value
11592           sta   t5
11594           jsr   mdr1        ;draw line
11596           ldy   #$0f        ;restore old
11598 mcd16     lda   mcbuf1,y    ; values
11600           sta   $a3,y
11602           dey
11604           bpl   mcd16
11606           ldy   t6          ;bump counter
11608           iny               ;value
11610           sty   t6
11612           cpy   #4          ;test counter = 4
11614           bne   mcd13       ; no
11616 mcd17     lda   $ab         ;copy ($ab)
11618           sta   $ad         ; to ($ad)
11620           lda   $ac
11622           sta   $ae
11624           inc   $a7         ;bump y counter
11626           lda   $ac         ;test hi val pos
11628           bpl   mcd19       ; yes
11630           eor   #$ff        ;complement
11632           tax
11634           lda   $ab
11636           eor   #$ff
11638           clc
11640           adc   #1
11642           bcc   mcd18
11644           inx
11646 mcd18     jmp   mcd20       ;skip
11648 mcd19     tax               ;set up .a, .x
11650           lda   $ab
11652 mcd20     stx   t2          ;store
11654           sta   7
11656           lda   $aa         ;test hi val pos
11658           bpl   mcd22       ; yes
11660           eor   #$ff        ;complement
11662           tax
11664           lda   $a9
11666           eor   #$ff
11668           clc
11670           adc   #1
11672           bcc   mcd21
11674           inx
11676 mcd21     jmp   mcd23       ;skip
11678 mcd22     tax               ;set up .a, .x
11680           lda   $a9         ;
11682 mcd23     sta   $14         ;store
11684           stx   $15
11686           lda   t2          ;abs hi of y
11688           cmp   $15         ;abs hi of x
11690           bcc   mcd24       ;x>y
11692           lda   7           ;lo of y
11694           cmp   $14         ;lo of x
11696           beq   mcd24       ;x>=y
11698           bcc   mcd24
11700           lda   $a9         ;copy current x
11702           sta   $ad         ; work value to
11704           lda   $aa         ; start value in
11706           sta   $ae         ; case of loop
11708           dec   $a8         ;dec x plot value
11710 mcd24     lda   $a8         ;test x<y plot
11712           cmp   $a7         ; value
11714           bcc   mcd25       ; yes
11716           jmp   mcd1        ; no - loop
11718 mcd25     rts
11720 ;
11722 mcmul     ldy   $a7         ;get y value
11724           jsr   mcmy        ;multiply by y
11726           stx   $1e         ; scalar and save
11728           ldy   $a8         ;get x value
11730           jsr   mcmy        ;multiply by y
11732           stx   $1d         ; scalar and save
11734           ldy   $a7         ;get y value
11736           jsr   mcmx        ;multiply by x
11738           stx   $1f         ; scalar and save
11740           ldy   $a8         ;get x value
11742           jsr   mcmx        ;multiply by x
11744           stx   $20         ; scalar and save
11746           rts
11748 ;
11750 mcmx      sec               ;flag - x
11752           .byte $24
11754 mcmy      clc               ;flag - y
11756           php               ;push flag
11758           jsr   $b3a2       ;conv y to f-p
11760           plp               ;retrieve flag
11762           bcc   mcm1        ;y multiply
11764           ldy   #>mxmult    ;point to x val
11766           lda   #<mxmult
11768           bne   mcm2
11770 mcm1      lda   #<mymult    ;point to y val
11772           ldy   #>mymult
11774 mcm2      jsr   $ba28       ;perform multiply
11776           jsr   $b1bf       ;convert to int
11778           ldx   $65         ;get value
11780           rts
11782 ;
11784 mctxt     lda   #0          ;duplicating off
11786           sta   t3
11788           jsr   mcg1        ;get x value
11790           stx   $a7
11792           jsr   mcget       ;get y value
11794           stx   $a8
11796           jsr   mcget       ;x size
11798           stx   $a6
11800           jsr   mcget       ;y size
11802           stx   $a5
11804           jsr   mcget       ;colour pattern
11806           stx   $b0
11808           jsr   $aefd       ;check comma
11810           jsr   $ad9e       ;eval expression
11812           jsr   $b6a3       ;make string ptr
11814           sta   $ad         ;store length
11816           ldy   #0          ;zero str counter
11818 mct1      sty   $ae
11820           lda   ($22),y     ;get character
11822           cmp   #$40        ;convert to
11824           bcc   mct2        ; screen code
11826           sbc   #$40        ; range
11828           cmp   #$80
11830           bcc   mct2
11832           sbc   #$40
11834 mct2      ldy   #0          ;calculate 16-bit
11836           sty   $ab         ; offset into
11838           ldy   #3          ; rom characters
11840 mct3      asl               ; by multiplying
11842           rol   $ab         ; screen code
11844           dey               ; by 8
11846           bne   mct3        ;address with
11848           sta   $14         ;upper 3 bits
11850           lda   $ab         ;add to char base
11852           clc               ; address $d000
11854           adc   #$d0
11856           sta   $15
11858           sei               ;lock out irq
11860           lda   1           ;switch out i/o
11862           and   #$fb        ; to see d-rom
11864           sta   1
11866           ldy   #7          ;copy 8 bytes of
11868 mct4      lda   ($14),y     ; char definition
11870           sta   mcbuf2,y    ; to buffer
11872           dey
11874           bpl   mct4
11876           lda   1           ;restore i/0
11878           ora   #4
11880           sta   1
11882           cli               ;enable irq
11884           lda   $a8         ;y ptr to plot
11886           sta   $b4
11888           lda   $a6         ;x multiplier
11890           sta   $a9         ;count down x val
11892           lda   $a5
11894           sta   $aa         ;count down y val
11896           lda   #0          ;reset row count
11898           sta   $ab
11900 mct5      lda   #7          ;init bit count
11902           sta   $ac
11904           lda   $a7         ;x value for plot
11906           sta   $b5
11908           ldy   $ab         ;get char row
11910           lda   mcbuf2,y
```

```
11912 mct6    asl   ;test left bit
11914         bcc   mct8      ; clear - no plot
11916         pha             ;push .a
11918 mct7    lda   $b0       ;get colour
11920         jsr   mcplo     ;plot
11922         inc   $b5       ;bump x value
11924         dec   $a9       ;dec x mult
11926         bne   mct7      ;loop till 0
11928         jmp   mct10     ;skip
11930 mct8    pha             ;push .a
11932 mct9    lda   #0        ;clear points
11934         jsr   mcplo
11936         inc   $b5       ;bump x val
11938         dec   $a9       ;dec x mult
11940         bne   mct9      ;loop till 0
11942 mct10   lda   $a6       ;reset x mult val
11944         sta   $a9
11946         pla             ;pull bit pattern
11948         dec   $ac       ;count down
11950         bpl   mct6      ;loop till <0
11952         inc   $b4       ;bump y value
11954         dec   $aa       ;dec y mult
11956         bne   mct5      ;loop till 0
11958         lda   $a5       ;reset y mult
11960         sta   $aa
11962         ldy   $ab       ;bump char row
11964         iny
11966         sty   $ab
11968         cpy   #8        ;test <8
11970         bne   mct5      ; yes
11972         lda   $a6       ;get x mult
11974         asl             ;times 8 for next
11976         asl             ; character
11978         asl
11980         clc
11982         adc   $a7
11984         sta   $a7
11986         ldy   $ae       ;bump index into
11988         iny             ; string
11990         cpy   $ad       ;test = length
11992         beq   a19       ; yes
11994         jmp   mct1      ;handle next char
11996 ;
11998 ropen   pha             ;open up ram
12000         sei             ; underneath the
12002         lda   1         ; roms, disable
12004         and   #$f8      ; the irq and
12006         sta   1         ; preserve .a
12008         pla
12010         rts
12012 ;
12014 rclose  pha             ;close up ram
12016         lda   1         ; underneath the
12018         ora   #7        ; roms, enable
12020         sta   1         ; the irq and
12022         cli             ; preserve .a
12024         pla
12026         rts
12028 ;
12030 add19   lda   $19       ;add screen width
12032         clc             ; (40, $28) to
12034         adc   #$28      ; pointer ($19)
12036         sta   $19
12038         bcc   a19
12040         inc   $1a
12042 a19     rts
12044 ;
```

```
12046 mctbl1  =     *         ;hi-res line addresses
12048 .word $e000
12050 .word $e140
12052 .word $e280
12054 .word $e3c0
12056 .word $e500
12058 .word $e640
12060 .word $e780
12062 .word $e8c0
12064 .word $ea00
12066 .word $eb40
12068 .word $ec80
12070 .word $edc0
12072 .word $ef00
12074 .word $f040
12076 .word $f180
12078 .word $f2c0
12080 .word $f400
12082 .word $f540
12084 .word $f680
12086 .word $f7c0
12088 .word $f900
12090 .word $fa40
12092 .word $fb80
12094 .word $fcc0
12096 .word $fe00
12098 ;
12100 mctbl2  .byte $3f,$cf,$f3,$fc
12102 ;
12104 mctbl3  .byte $1d,$1d,$1e,$1e
12106 ;
12108 mctbl4  .byte $1f,$1f,$20,$20
12110 ;
12112 mxmult  *=*+6
12114 ;
12116 mymult  *=*+6
12118 ;
12120 mxtype  .byte 0,1,0,1,0,1,0,1
12122 ;
12124 mytype  .byte 0,0,1,1,0,0,1,1
12126 ;
12128 mxadds =     *
12130 .byte $1d,$1d,$1d,$1d
12132 .byte $1e,$1e,$1e,$1e
12134 ;
12136 myadds =     *
12138 .byte $1f,$1f,$1f,$1f
12140 .byte $20,$20,$20,$20
12142 ;
12144 mcbuf1 *=*+$10
12146 mcbuf2 *=*+8
12148 ;
12150 *       =     *+(*&1)   ;skip odd byte
12152 evtmp   .word $e38b      ;error vector
12154 ;
12156 mcfxer  sei
12158         lda   $300      ; vector to temp
12160         ldy   $301      ; storage
12162         sta   evtmp
12164         sty   evtmp+1
12166         lda   #<mcerr   ;copy substitute
12168         ldy   #>mcerr   ; address to page 3
12170         sta   $300
12172         sty   $301
12174         cli
12176         rts
12178 ;
12180 mcerr   jsr   mcrstr    ;restore lo-res
12182         jmp   ($300)    ;handle error
12184 ;
12186 mcuvid  *=*+3           ;store user video
12188 ;
```

Program 5: INLINE

```
PA  0 rem inline (aug 25/84)          :
FH  1 :
AI  2 rem  1 statement, 0 functions
HH  3 :
GO  4 rem keyword characters: 6
JH  5 :
NJ  6 rem keyword      routine    line     ser #
CB  7 rem inline       inlin      3454     030
MH  8 :
HD  9 rem ================================
OH  10 :
MM  107 .asc "inlinE"
OA  1107 .word inlin-1
GJ  3454 inlin    cmp #$22      ;test quote
DF  3456          bne inl1      ; no
BL  3458          jsr $aebd     ;eval prompt string
NN  3460          lda #$3b      ;check semicolon
OC  3462          jsr $aeff
OK  3464          jsr $ab21     ;print prompt
DN  3466 inl1     jsr $b3a6     ;check prg mode
LB  3468          jsr $a560     ;get input
JE  3470          stx $43       ;store ptr to input
OL  3472          sty $44
LJ  3474          lda #0        ;find/create var
DM  3476          jsr $b08b
EO  3478          jsr $ad8f     ;check string type
BM  3480          sta $49       ;store pointer to
EG  3482          sty $4a       ; descriptor
EA  3484          lda $7a       ;save chrget ptr
OL  3486          ldy $7b
IJ  3488          sta $4b
NP  3490          sty $4c
NC  3492          ldx $43       ;set cg ptr to
NG  3494          ldy $44       ; start of input-1
PP  3496          stx $7a
IA  3498          sty $7b
OA  3500          jsr $73       ;bump cg ptr
MJ  3502          lda #0        ;set str delimiter
JC  3504          sta $07       ; to zero
EI  3506          sta $08
CI  3508          lda $7a       ;set up string for
DM  3510          ldy $7b       ; assignment to var
FP  3512          jsr $b48d
IM  3514          jsr $b7e2     ;reload cg ptr
LO  3516          jsr $a9da     ;copy str to var
DI  3518          lda $4b       ;restore cg ptr
NN  3520          ldy $4c
NL  3522          sta $7a
CC  3524          sty $7b
CL  3526          rts
OD  3528 ;
```

The Amiga: A User's Perspective

Chris Zamara, Technical Editor

The Amiga provides new answers to the old question, "Great, but what's it good for?"

The Amiga may fill the promise of being the tool for the average computer-naive person and enthusiast alike. Because of its visually oriented user interface, the consistent way of doing things among all programs, and the power and capabilities inherent to the hardware itself, this eye-opening machine can be used and enjoyed even by the most computer-shy among us. And not used as a toy, but as a flexible tool for persons in many fields. It is not a business computer, or a "home" computer, or a music computer, or a graphics computer, or a software development system. It is all of these, and manages to be well suited to both the computer-phobe and computer-phile.

Regardless of which of these categories you fall under (I strongly expect the latter, though), the following is a look at the main elements of the Amiga from the end user's standpoint. (The machine is also great for programmers, but that's another article.) The elements of the system which concern the user are: The WorkBench, Amiga's efficient and intuitively-operated user interface; the bottom line – examples of actual application software; and the basic hardware in terms of ergonomics and expandability. What follows is a closer look at these three elements of the Amiga system and philosophy.

The WorkBench: Your Link With The System

The Work Bench is a very fun, friendly and modern (read: Macintosh/GEM -like) front end to the powerful AmigaDOS operating system lurking beneath. It lets you control the tasks and files in the system without knowing that you're doing such ominous-sounding work. You'll be startings tasks and copying files, but as far as you're concerned, you're just using the mouse to point to things and select them, or "drag" them over to where you want to put them.

When workbench first comes up (after you boot the system), all you see is a little picture of the disk you currently have in the system drive with its name underneath (if there is more than one drive, all mounted disks will be displayed). Point to the disk with the little mouse-controlled arrow on the screen, then click the left mouse button twice, and ZAP – a "window" appears on the screen displaying the contents of that disk. The programs on the disk (called tools) are displayed as icons – little descriptive pictures – which can be double-clicked like the disk icon to run the program. A window may contain *drawers*, which are sub-directories containing several tools, projects (files used by tools), or more drawers. Opening a drawer brings up another window somewhere on the screen, showing the contents of this new drawer.

Now it may sound like quite a mess with all of these drawers, windows, icons and program names floating around, but each window has various *gadgets* which let you work with only what you're concerned with. Using the gadgets, you can change a window's size by dragging its right bottom corner, and its location on the screen by using the "drag bars" to move it wherever you wish. You can also push a window to the front or the back of a stack of windows using another pair of gadgets on the window. All of these operations are done by simple mouse moving and clicking. If a window is too small to see all of the icons within it, you can either enlarge the window or use the scrolling gadgets to see different parts of what's in a window. A "close" gadget allows you to get rid of the window. In words, all of this may sound rather confusing, but it literally takes only a few minutes to become comfortable with the whole operation.

The icons representing the tools, projects and drawers in any window can be dragged around to other windows. Doing so will copy the file to whatever disk or drawer you drag it to. You can duplicate an entire disk by dragging its icon to the icon of another disk. To get rid of a file, you just drag it into the Trashcan, an icon in the main disk window. Until you empty the trash, though, you can still retrieve trashed files by opening the Trashcan. All of this is great fun, and makes you feel like re-organizing all of your files just for the heck of it. Besides copying files and running programs, the WorkBench has *pull-down menus* which allow other functions, like renaming, disk initialization, emptying the trash, getting information about a file, drawer or disk, and "cleaning up" a window to neatly organize the icons contained within it.

Pull-down menus are an important aspect of Amiga's environment, used by application programs as well as workbench. Pressing the right mouse button reveals menu titles, and pointing to one of these menus pops up a list of options under the menu name. These options can be selected with the pointer, and some may have sub-menus which pop up to the side, containing additional choices. For example, a "change color" menu option may have a sub-menu containing the choices red/green/blue.

When a tool is opened (running a program), it can run in its own window in the workbench. For example, you may open a terminal package which brings up a new window. This window acts as a computer in itself, with the terminal program running inside. Depending upon the tool running, this window can usually be re-sized, re-ordered, or closed just like any other window. You are free to start as many programs as you wish in this manner (limited by available memory), each with its own window; and **each running at the same time!** By clicking any given window, you "select" it so that you can supply input to its program and activate that program's pull-down menus. The ability to have a number of programs running, each available at any given moment, adds a new dimension to computer usage. Consider this: you're writing an essay on a word processor, and while it's performing a long search through text, you get bored and click your adventure game to the front. Just before you're about to bite the dust at the hands of a bloodthirsty troll, you notice that someone has called up your Bulletin Board System, which is running in its own window, barely visible above the troll's misshapen head. After enlarging the BBS window and chatting with the fellow who logged on (the troll has meanwhile enjoyed a satisfying lunch), you decide to have a look at how much room is left on your disk, so you check the INFO menu after clicking the main workbench screen, visible under layers of windows. Clicking back to your essay, you find that your word search has long since completed and you are ready to write on. A fanciful example perhaps, but there are many real-world occasions when you want to do something with the system or another program without having to save your project, bring in a new program, then bring in the original program again later. Just the ability to easily switch between tasks is a marvelous convenience; multitasking as well means we're talking serious computing power here.

Some programs take up an entire screen rather than a window. This doesn't mean that it dominates the machine, though; you can slide the screen down to reveal whatever was there before. Like windows, more than one screen can be active, and each can be slid up and down to reveal whatever is desired. Like windows, screens have re-ordering gadgets to push one to the top or bottom of a stack of screens. For a major application program, or one that needs a constant screen width, a screen can be more convenient than a window. Each screen may have windows within it, since a program may make its own windows.

Using workbench gives you an idea of the Amiga's software philosophy. You can accomplish a lot without ever using the keyboard, and you don't have to know a thing about computers to get things done. You'll find that your work on the computer is organized more like your daily tasks, because you can schedule your time so that several things may be going at once. The workbench screen will probably be as neat or as messy as you keep your desktop. Once you can use the mouse to control workbench functions, you shouldn't have any trouble using application software – the basic principles of pointing, clicking and menu selection apply to most programs that run on the Amiga. Let's now take a look at some of these programs.

Application Software

One of the ideas behind the Amiga that makes it so easy to use is that once you learn to use the mouse to point, click and choose menu options, you can operate most software. On a Commodore 64 or similar computer, there is no such thing as "learning how to work the computer"; every program that you run will have its own rules and conventions for choosing options or selecting functions. With the Amiga, software developers are encouraged to use the routines in the operating system and to follow certain conventions. This means that you can bring up a program and start using it right away because you know how to access the menus; how to save, load, or edit a project; how to select any options displayed on the screen, etc. A consistent user interface among different programs is truly a wonderful thing!

Besides the fact that programs are generally operated in the same manner, they also will (at least they *should*) work with other programs in the system, in a screen or window which can be pushed aside to work on something else. This last point is a convention rather than a rule, since it is possible to write a program which hogs the whole machine and requires that the program be quit, or even the system re-booted, if you wish to use another program. Such system-hogging programs, however, totally defeat the purpose of the Amiga's remarkable user environment and should be avoided. Before you buy a program, make sure that it can allow the Amiga to be used as intended; as a multi-tasking machine. Avoiding the purchase of a system hogging program will serve two purposes: 1) save you much frustration when you discover that you can't use it with other software, and 2) hopefully establish an unwritten law for software developers governing all future Amiga programs. While I'm beefing, one more potential problem with Amiga software: if you have external RAM on your system in addition to the internal 512k, some programs will not run properly because they don't specifically allocate video memory when they need it (the video chips can only access the internal 512k). This hopefully won't be a continuing problem, as long as software developers test their programs on systems with external RAM.

In order to get an idea of the kind of products you can expect for the Amiga, below is a look at three programs currently available, covering different aspects of the Amiga's capacity: graphics, word processing, and music. These are not product reviews, just impressions of programs to illustrate typical Amiga applications.

Deluxe Paint From Electronic Arts

Deluxe paint is a real showcase for the Amiga'a colour graphics, and probably the most powerful graphics package on any microcomputer. It is oriented towards creating works of art rather than design applications, and has a mind-boggling array of ways in which to manipulate images. Just a few are given below. Besides the usual line, box, ellipse, and paint commands, there are unusual features such as being able to scale any area's width and height in real time, by moving the mouse.

Rotation of any boxed area is also provided for. The 32 on-screen colours can be set up to encompass any of the Amiga's 4,096 colours by using a palette control which lets you set the amount of red, green, and blue for every colour. The paint palette can also create a range of desired colours between any two colours you choose, for example shades of grey between white and black. You can choose a cycle mode which cycles through the colour registers to give the illusion of movement in various parts of the picture. The magnify mode makes it easy to work in detail on any small section of the picture, and different magnification levels can be chosen. there is an airbrush mode which works like a real airbrush, spitting out "drops" of the desired colour randomly over a given area. A shading feature allows you to make anything you paint over a shade lighter. You can define any part of the graphics screen as your "paint-brush" and use it for any future painting. This package is a natural for the Amiga, using the 4,096 colours for visual effect, the custom chips for super-quick area fills and shape drawing, and Intuition (the operating system) for an easy user interface.

Although DPaint is an extremely complex package, using it is quite simple because it follows the conventions for Amiga programs. Main functions are selected from pull-down menus, and modes can be switched by pointing to the desired icon and clicking, though some of the icons are a bit cryptic until you figure them out. The program runs in a standard screen, which means the whole thing can be pulled down or re-ordered (by clicking the re-ordering gadget) to reveal WorkBench or what-ever was up before. Good! The program comes on a copy-protected disk. Not so good, because you can't put the program on you favorite utilities disk or make backups. I hope it isn't necessary for future Amiga software to be copy-protected, but if Commodore-64 type piracy persists with the Amiga, it may be. Sigh. Incidentally, DPaint is selling for a reasonable price considering the quality and power of the package – let's hope this sets a trend!

TextCraft Word Processor From Commodore

Textcraft is a very easy to use (there it is again, but it's still less over-use than "user-friendly"), "WhatYouSeeIsWhatYouGet" word processor. The nice thing about Textcraft is that there are *no* text-formatting commands to learn. It takes full advantage of a mouse-driven system. Change the margins? Grab the little margin markers and move them to where you want – the current paragraph re-formats automatically. Change the text format? Click the desired icon to instantly re-format. Headers, footers, page length and the like are selected from requesters brought on-screen by a menu option. Text editing functions are performed by the current pointer you're using: pencil to add text, scissors to cut, camera to copy, glue bottle to paste, paintbrush or roller brush to change fonts or text formats over a given range. Deleting a section of text is easy: just get the scissors, drag them over the text you want to delete, highlight-ing it, and release the mouse button. In the blink of an eye the text is gone, and may be pasted elsewhere in the document with the paste icon if you wish. You could figure out everything about using TextCraft just by playing, but even so there is extensive help available from a help menu, even providing one minute tutorials on every facet of the package's operation. Forgive me for repeating this just one more time: very easy to use.

As you edit, text on the screen appears exactly as it will on the printed page, including italics, bold-face and underlined text. You don't have to guess if *this* printout will finally be the right one. Textcraft is so unintimidating that it will pass this software test: take your favorite computer-phobia victim, you know, the one who is scared to get near a computer, let alone touch it, since he is certain that his smallest action will result in the instant destruction of the frightfully expensive system, or worse, that his every move will be electronically recorded and surreptitiously passed on to an evil organization who will gather information from computer-naive people and go on to spread darkness and evil throughout the free world. Take that person, sit him or her down in front of Textcraft, and clench his white, clammy hand around the mouse. Now tell him he has the opportunity to write that letter he's been putting off for years. Providing he's used a typewriter keyboard before, the chances are good that colour will slowly come to his mouse hand, a letter will take form on the screen, and a new mouseke-teer will be born. The power of easy software!

The down side is that Textcraft does not have the slew of advanced features found in some other word processors. For things like spelling checking, column manipulation, sorting, virtual memory, automated table-of-contents, or filling of variables from a file, you'll have to look elsewhere. Lots of features though, does not preclude ease of use, and soon there may be a Textcraft look-alike with a plethora of features. The point is, word processing isn't the old, "Hmmm . . . was that a Control-Big-C or Control-Small-c to set a column? Or was it a Control K? I Wonder what this mess is gonna *look* like?. . .". Thank God.

Incidentally, the above observations are based on a pre-release version of Textcraft that I saw, but I believe the release version has been very little changed. The version I saw provided no obvious way to re-order or slide its window to get to other programs in the system. Horrors! I hope this has been corrected in the version now for sale.

Musicraft from Commodore

Musicraft is an incredible music composition tool that really lets your musical creativity express itself, even if you can't play an instrument. You use the mouse to put notes up on a staff, composing the music as it would look on paper. Notes can be changed, inserted and deleted. Music for up to four voices can be composed, any combination of voices being displayed or played at once. The exceptional thing about Musicraft on the Amiga is the instruments available. The realism is unbeliev-able: drums sound like *drums*. If you've heard computer synthesized drums before, that's really saying something. The same goes for other hard-to-produce sounds like bells, electric guitar, and banjo. There is a large assortment of instruments on the Musicraft disk, and each sounds eerily realistic. The trick, of course, is Amiga's approach to sound; in effect, the instruments

are digital recordings stored in memory and converted to analog at a high rate by DMA. That gives you something like a programmable tape recorder to play with, making complex sounds a piece of cake. You can change instruments at any time in your composition simply by putting an instrument icon on the staff instead of a note or rest. Editing a song is fast and simple, and while it's playing, you can adjust the speed, tempo and other settings by "sliding" on-screen "controls" with the mouse-driven pointer. There are some pretty impressive sample songs on the disk that you can bring in to play and edit.

Besides staff mode, as it's called, there is also synthesizer mode. The synthesizer is a screen full of controls which you can use to create your own instrument to use or save on disk with the others. You can actually draw your own waveform on a little graph, or generate a waveform mathematically by choosing a standard function and adding two or three times its frequency to itself at any amplitude. There are literally dozens of mouse-operable slide controls on the synthesizer, which control filters, oscillators, amplitude generators, and various levels. The set-up basically simulates a good synthesizer, but even if you don't know what half the stuff does, you'll have a ball just messing around with the controls and listening to the results (take it from me). Musicraft would also pass the test by your local computer-paranoid, but this one doesn't even have to know how to type. In fact, the keyboard isn't required at all, unless you want to play notes live, or play along with one of your compositions. A keyboard screen lets you see what notes are assigned to which keys, and lets you change the assignments. In the case of Musicraft, its simple operation is more than just a convenience, since it opens the door for many musicians who have no desire to learn about programming computers or how to use a complex package. I have seen a non-computerist guitar player sit down in front of Musicraft and start producing music after a few minutes, concentrating on the composition itself rather than on how to use the program. That's how it should be.

The Amiga Hardware

Here's something that Commodore 64 users in particular will appreciate: to add a standard Epson-type printer to the Amiga, you plug it in and it works. Period. Forget the interface. There is a standard parallel printer port provided on the back, and the operating system supports most popular printers and can use their special features. You can tell the system what kind of printer you have using the "Preferences" tool on the workbench disk. There is also a serial port on the back, but it uses a male instead of the standard female connector, requiring a special cable or a male-female adapter. Other than that, it is a standard RS-232 port, which you can use to connect a modem for telecommunicating. The serial port can run at speeds up to 19,200 baud.

The computer unit itself sits on supports on either side and leaves enough room underneath for the keyboard to slide under when not in use. Handy when desk space is prime real estate. Speaking of the keyboard, it is quite light and sits nicely on the knee. The fan in the main unit is so quiet that I'll bet there are some Amiga owners out there who are reading this and saying, "What fan?".

The built-in disk drive uses 3.5 inch microdisks. These disks are much better than floppies since they are protected by a hard plastic case and require no storage sleeves, since the recording surface is protected by a sliding trap which is moved away when the disk is in the drive. Each disk holds around 880K, and data access is quite fast, a disk duplication taking only about 90 seconds. Perhaps best of all, the disks fit in a standard shirt pocket. If anyone actually took that detail into consideration when designing the 3.5 inch format, I tip my hat to him.

For further expansion of the system, there is a slot on the right side of the machine; devices can be stacked horizontally out to the side of this port. This doesn't seem as elegant or reliable as a card-cage like in the IBM PC, but the Amiga would have probably had to grow too much in size and cost to accommodate such an arrangement. Several manufacturers will be offering motherboards though, to allow proper expansion of multiple boards. You can also add more memory. Several outfits, including Comspec Communications here in Toronto, will be offering two megabyte RAM expansions as soon as additional information about Amiga's I/O protocol is released to developers. I've been using one of Compsec's prototype RAM expansion units for a few weeks now, and having all of that memory can really speed things up when you use it as a super-fast disk drive. Hard drives are also here now or on the way soon from various companies. Imagine a 2.5 Megabyte Amiga with a 20 Megabyte hard drive! Awesome, as Commodore's promotional people say.

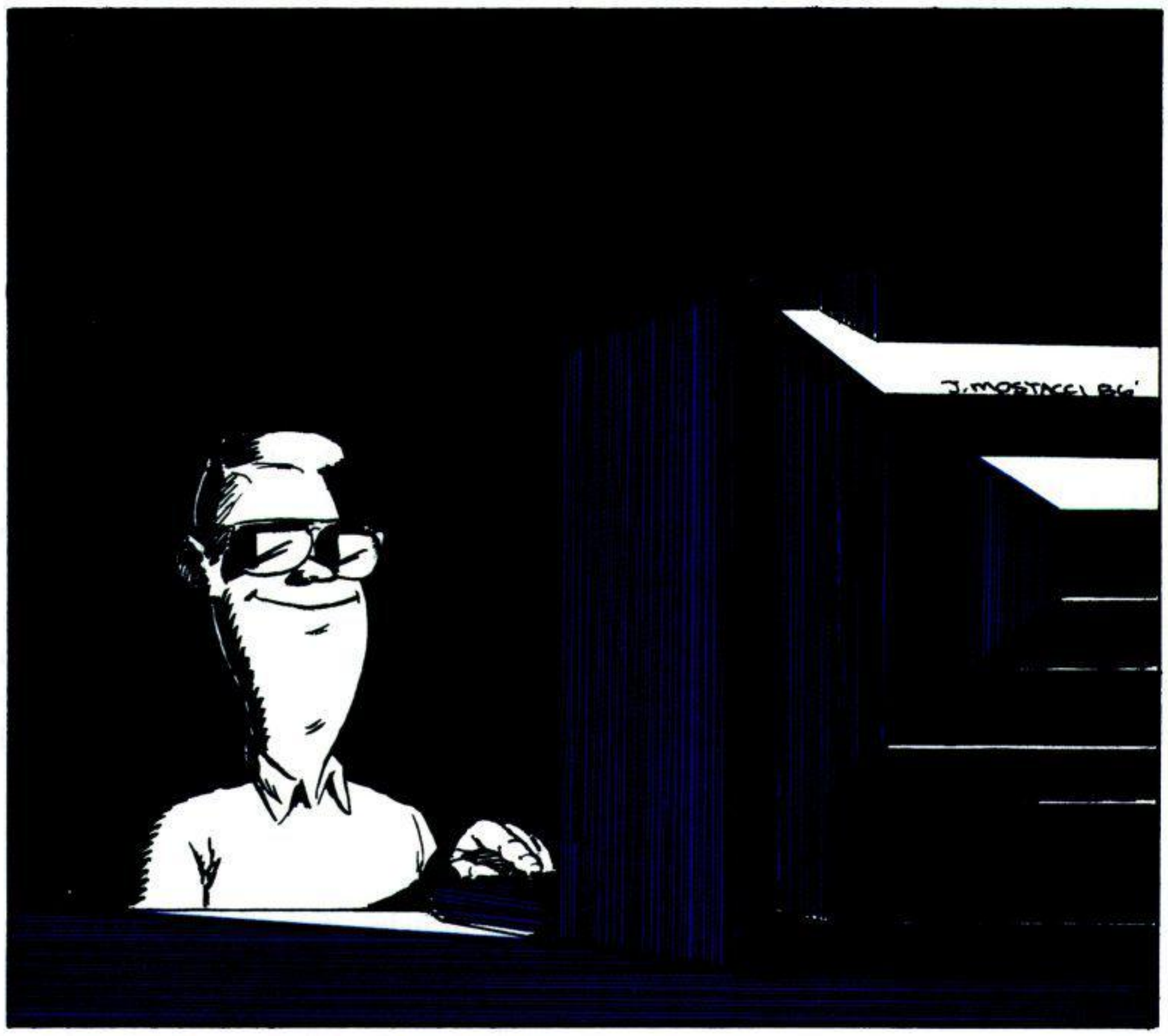
Is It Worth it?

Judging from the Amiga's real-world usefulness when running available, affordable software, it would not be an exaggeration to say that the Amiga may well be worth its price just to run one specific program. Musicraft, for example, would be reason alone for a budding musician to buy an Amiga. Likewise Graphicraft for the graphic artist or Textcraft for the writer. But best of all is that anyone can make use of the machine regardless of his field of interest or level of computer expertise. I just can't see an Amiga languishing in anyone's closet like some of the simpler eight-bit machines have been known to. The Amiga appears to be the first micro to combine multitasking, an easy and flexible user interface, advanced graphics and sound hardware, and a good operating system. Its only real competition is the Atari 520ST and Apple's Macintosh, which are good machines in their own right, but neither are multitasking, both lack the super-fast hardware graphics capabilities, and the Mac doesn't have a colour display and isn't easily expandable. Which machine wins out remains to be seen, but the Amiga is a very strong contender and deserves your consideration if you're looking around. No machine is without its flaws, but the Amiga has enough virtues to convince many that it's the best micro available . . . for now, anyway.

The Amiga: A Programmer's Perspective

Chris Zamara, Technical Editor

. . .can a single mortal whip up a game or utility within a normal lifetime?



The Amiga is designed to be powerful yet easy for anyone to use. In other words, it is a machine oriented to the user, with much of the application software being developed by the major software companies. But how is it for the average hacker – can a single mortal whip up a game or utility within a normal lifetime?

Well, he'll have a bit of learning to do, but using one of Amiga's BASICs, he'll be writing programs using multiple windows, graphics, pull–down menus, the mouse, and even speech in no time. All of those goodies – and others – which are inherent to the Amiga environment are available to the programmer even in Assembler, because of the built–in operating system functions and the libraries available on disk. Programming at the lowest level, i.e. manipulating the hardware itself, is pretty much out, but the operating system does so much for you that you'll never need to do so.

The Amiga comes with BASIC (MetaCompco's ABasiC or Microsoft's AmigaBasic), and at the time of writing, "C", Pascal, and Assembler packages are all available at dealers. The developer's kits which are available for the Amiga contain complete system documentation and manuals for the included Lattice "C" compiler and the 68000 Assembler – seven manuals in all. Most of the elements of this kit are now, or will soon be, available at Amiga dealers. The manuals will be released to the public as soon as they are finalized; fortunately, the folks at Commodore–Amiga seem extremely willing to make the information widely available! With the help of these manuals, all of the Operating system's features like windows, menus, requesters, etc. can be used from C or assembler; the documentation is complete and keeps no secrets about the system. Perhaps the only "bad" news for hackers is that there are no mysteries to solve or unknown routines to find!

Since the idea behind the Amiga is that anyone can use it intuitively (hence "Intuition" as the name of the operating system), there are certain standards that all programs must follow to work properly in the machine, and other standards which are recommended so that there is some consistency among application programs. These standards and the many Kernel routines available make a bit of study necessary for the new Amiga programmer, but allow the creation of complete, friendly, professional programs.

For many applications, programming in either of the available BASICs is an extremely easy way to realize much of the Amiga's potential. If you want to use C or assembler, you have even more control over the system, but you'll need documentation, like the Intuition and ROM Kernel manuals, to use all of the system's features. These manuals are currently available only in the developer's kits, but should be at Amiga dealers soon, probably by the time you read this.

Documentation

As previously mentioned, the Amiga people are keeping few secrets about the machine, and there doesn't seem to be a lack of information as there was, for example, early in the Commodore PET days. Some of the manuals aimed at developers are a

bit cryptic and extremely technical, but the ones aimed at the user – like the AmigaDOS User's manual – are very easy to understand and full of examples. Since the Amiga is a very new and complex system, updates on these manuals are occurring constantly, and the stack of errata pages included with the developer's kit is quite large. When the final documentation is actually released to the public, the manuals should be fairly accurate, as the questions of developers serve to weed out the errors and omissions.

Workbench and BASIC manuals come with the system, and for the casual BASIC programmer contain everything he'll probably need to know. Workbench is Amiga's Icon–oriented user interface, and the manual explains how to use it to copy, rename, and delete files, duplicate disks, run programs ("tools"), use the window gadgets, etc. Workbench is designed to make the system intuitively easy for users to operate, but all of its functions can also be performed directly using AmigaDOS from CLI, the Command–Line Interface. If you wish to fully understand and use CLI and the DOS commands, the set of three AmigaDOS manuals is available from dealers. Other Amiga books from outside publishers are expected to arrive shortly as well.

The amount of documentation already available suggests that the Amiga will be a very "open" system, and software will abound from independent users everywhere. In addition, a dedicated magazine, Amiga World, already exists. This is no shoddy newsletter, but a high–quality, full colour publication. Sources of Amiga information for programmers also include networks with an Amiga section such as UseNet, not to mention the Amiga–specific BBS's, User Groups, and newsletters which are springing up around the continent. The Transactor and other Commodore–specific magazines are also covering the Amiga at one level or another.

AmigaDOS and the CLI

Even without bringing a particular language into the system, you can do some useful things using AmigaDOS through CLI. The CLI (Command Line Interface) is a process provided by AmigaDOS, and can be run from WorkBench. Like WorkBench, CLI is a user interface to the system, but is command–rather than icon–driven. It simply reads commands and executes them. More than one CLI can be active at once, each running in its own separate window; a new CLI can be started with the NEWCLI command, and a CLI can be ended with ENDCLI. Actually, when a WorkBench disk is booted, CLI comes up and executes a batch file – a series of DOS instructions – which invokes WorkBench with the "LoadWb" command. In other words, CLI is the first thing the system runs, and it is quite possible to run the machine solely from CLI and never use WorkBench at all.

DOS is used primarily for disk management and I/O commands, and allows you to execute commands or any program on disk by simply typing its name. A disk's main or "root" directory may contain files, or other directories which may contain other files or directories, and so on. DOS maintains a disk directory hierarchy and allows you to work with any file or directory on disk. Some of the things you can do with DOS: display any or all directories, sorted or with the date/time of creation and size also displayed (DIR and LIST), print files to the screen as text or hex (TYPE), COPY, DELETE, RENAME or JOIN files, set or view the DATE and time, and similar file–handling and general system functions. The RUN command causes any command or program to execute under a new CLI, allowing you to continue in the current CLI while the new job executes; in other words it lets you multi–task. Two standard commands are ED and EDIT, which invoke screen or line editors. For complete DOS, ED and EDIT command lists, see articles elsewhere in this issue.

All of the DOS commands are on disk as separate, executable files, and exist in the "C" directory on the workbench disk. When a command is called by name, it is fetched from disk and executed. This arrangement makes it possible to add, change, or rename commands at will. The number of commands you have available in DOS is only determined by what's in your "C" directory, and you can create you own commands simply by writing a program and putting the object file there.

Any input or output of a DOS command that would normally go to the current window (keyboard and screen) can be re–directed to any file or device that you wish, using the < and > operators. This way, you could, for example, send output from a command such as DATE to a file, or send information to a printer on the serial port instead of the screen. You can actually create a custom window of any size from the CLI and use it for input or output, as you would a disk file or device. Another useful device is the built–in RAM disk, which can be used as a disk drive; you can copy to or from RAM, get a directory listing, make sub–directories, even execute programs or DOS commands out of RAM. The RAM device is quite handy for program development, since you can speed up compilation or assembly enormously by doing it directly from memory. This is especially attractive if you have additional expansion RAM on your system.

CLI has a provision for batch files – sequences of DOS commands which are executed directly from a text file. Batch files are like simple programs written in the language of DOS, and even support primitive control structures: IF . . . ELSE . . . ENDIF, and SKIP (a forward GOTO). Batch files can also accept parameters when executed, giving a kind of "meta–command" ability, a command which consists of a series of other commands. Batch files are normally invoked by the EXECUTE command, but an exception is the file called "Startup–Sequence" in the "S" directory, which automatically executes when the disk is booted. "Startup–Sequence" normally prints a few things, then does a LOADWB to enter WorkBench and an ENDCLI to kill the CLI process, leaving you completely under the control of WorkBench. You may use Ed or EDIT to change

the Startup-Sequence, making the system leave you in CLI after re-booting, so that you can use DOS directly without having to invoke a CLI from WorkBench. You may set up your Startup-Sequence to do anything you want, like display the date, copy your favourite commands to the RAM disk, start a given program, etc.

ABasiC: The Original Amiga BASIC

ABasiC from MetaCompCo is the BASIC that until recently was shipped with every Amiga. It is in many ways an old-fashioned BASIC, much like that on a Commodore 64, but with many more commands in its vocabulary. This BASIC uses line numbers, GOTOs, no indenting of program lines, and LISTing of programs by line number just like in the old days. It does support a WHILE . . . WEND control structure, but that's about it for modern features. In fact, in the editing department it's back to the dark ages when teletypes roamed the computer rooms of the earth: it is a line-oriented editor. No moving the cursor over a line and simply changing it to correct it. At first this can be a disappointment to those weaned on Commodores or similar screen-oriented systems, but once you get used to the editor's commands, it's really not all that terrible.

The language itself is very rich, allowing mouse input, the creation of real windows with all the usual gadgets, a complete set of graphics instructions (which work in the blink of an eye thanks to the Amiga's custom graphics chips), speech and sound capabilities, error trapping, and just about anything else you could ask for function-wise. File handling is quite sophisticated, and works in a similar manner to IBM PC's BASIC; random file records are mapped to string variables with the FIELD command. Windows are treated as files and can be PRINTed to and INPUT from. All graphics and mouse input parameters are always relative to the currently active window, allowing the user to move around any windows that a program is using without the program even knowing – unless it wants to. Debugging instructions include TRACE, and a wonderful command called FOLLOW which allows you to trace any variable and observe it whenever it changes value. The commands of ABasiC cover such a wide range of functions that there's no way to list them all here. Suffice it to say that as far as commands and functions go, this language is not wanting.

ABasiC checks every program line for syntax as it is entered, pointing out where in the line an error occurred. Programs are stored as pure ASCII files, and can be edited with you favorite editor, like ED from CLI, if you wish. This form of storage also means that programs are not particularly compact or fast. In floating point operations, ABasiC on the Amiga is not much faster than some BASICs running on lesser machines like the IBM. When it comes to graphics though, stand back!

When ABasiC comes up, it creates a new screen. It actually runs in a window which happens to take up this new screen. That means that by re-sizing the ABasiC window, you can get to the screen behind it and slide it down to reveal WorkBench happily waiting for your return. You are free to merrily switch between BASIC and WorkBench without interrupting any task which may be executing, like a BASIC program. It's very handy to have access to a CLI at any time without having to abandon your current program, and even see part of the program on the screen. You can even bring up another BASIC, and have both of them running in their own screens, letting you slide them around and work with whichever one you wish. You can bring up as many BASICs as you have memory for, which isn't many unless you have more than the internal 512k of RAM. The default ABasiC screen is a 320 by 200 two-bitplane screen which allows for 40 characters per line. With the SCREEN command, you can change to a 640 by 200 screen and get 80 columns.

You can definitely write some interesting programs using ABasiC, and for many applications it will be fast enough. But since it *is* just an ordinary BASIC, your code is liable to be difficult to debug, as all variables are global (no passing of parameters to subroutines), and all the necessary GOTOs pointing to number by line can weave a tangled spaghetti web. For major applications, things can get just a little bit too messy, but if you're used to that sort of language, then you'll like it just fine. Other than the line editor, it is far superior to the built-in BASIC on any previous Commodore.

AmigaBasic From Microsoft

Microsoft's AmigaBasic is the replacement for ABasiC and is being shipped with all new Amigas. If you bought an Amiga with ABasic, see your dealer about getting an upgrade kit to AmigaBasic. AmigaBasic is an up-to-date language which is powerful, fast and completely structured. Program line numbers are not required, and your program is edited in a separate window from the main "run" window, which displays program output and lets you enter direct-mode commands. The WHILE . . . WEND and IF . . . THEN . . . ELSE . . . ENDIF constructs exist, and real sub-procedures can be used which are invoked by name and can be passed a list of local variables or arrays. In short, this BASIC is a structured, COMAL-like language which is interactive, powerful, and easy to program in.

AmigaBasic supports most of the commands of ABasiC, with a few new features and some new twists. The major advantage that this BASIC has is its "Event Trapping" capabilities. The ON MOUSE command, for example, can be used to name a subroutine to be performed when the mouse button is pressed. The program then need not check the mouse repeatedly, but the mouse-handling routine will be automatically executed when necessary. The ON MENU command is used for checking pull-down menus, which were not supported in ABasiC. When a menu option is selected, you can have your menu-handling procedure performed, even though your program is not explicitly checking for a menu action at the time. These interrupt

capabilities exist for object collisions, timer countdowns and program breaks as well. Windows are supported more fully than in the previous BASIC; now you have a choice as to what gadgets you want on any given window. Object animation is supported, and an object editor comes on the AmigaBasic disk to create sprites and "BOBs" (software sprites). You can directly call system library functions if you open the library with the LIBRARY command. Again, there isn't enough room to list all of the available functions, but there are enough. The beauty of this BASIC is that you can write your own procedures, much like in COMAL, which are then used exactly like the built-in commands. In other words, your program defines new commands for you to use and build upon.

The editor is full-screen, in a window of its own. You can scroll up and down through the listing in this text editor of sorts. The mouse is used to place the cursor at any spot and to highlight sections of text for cut/copy/paste operations, or replacement of new text. Just use the mouse to highlight any text in you program, and type the new text to replace it. Programs are totally free-form in this editor, allowing blank lines anywhere and indenting of lines to emphasize the control structures.

Unlike ABasiC, the AmigaBasic environment takes full advantage of the Amiga's nature, using the mouse, windows, pull-down menus, and Amiga-Key "shortcuts". In fact, it gets a bit carried away and forces you to use the mouse for no good reason: any program error MUST be acknowledged by clicking an "OK" box in the error requester before continuing – a bit of a pain. Also, updating the screen after one of these errors, or sometimes while editing, is a bit slow. The environment is not hacker-oriented, as it puts ease-of-use over speed, but it sure is flexible.

AmigaBasic occupies not a screen, but two windows in the WorkBench environment – one for listing/editing, and the main window for issuing commands. These windows are no different from any other, and can be modified by the WINDOW command at will. Since AmigaBasic programs can be executed from WorkBench (they have their own icon), you can have a program change the main BASIC window and come up as an ordinary tool – the user never has to know it is a BASIC program.

Programs are semi-compiled and stored in a compact way. Keywords are tokenized and variables are stored not by name, but by reference, with all of the variable names stored at the end of the program file. This saves memory and speeds things up considerably; standard benchmark programs seem to run about three times faster in this than in ABasiC. You may optionally save a program as straight ASCII if you wish, or save it protected so that it can't be modified.

All in all, AmigaBasic is a good programming environment and can be enjoyed even by those who scoff at normal BASICs. If it's still not fast or flexible enough for your application though, you'll have to go to C or assembler.

Lattice C Compiler For The Amiga

The Lattice C compiler is part of the developer's kit, or available as a separate package from Amiga dealers for a lot of money (retail about $450 in Canada). I will not attempt an explanation of the C language in general, but will just give some notes on Lattice C on the Amiga. If you aren't a C programmer and wish to learn about this widely used language, the definitive resource is the book, *The C Programming Language* By Brian W. Kerningham and Dennis M. Richie.

First of all, you need at least 512k and two drives to use the C development system. It is apparently possible to use a single drive for C development, but it is probably quite inconvenient. Lattice C on the Amiga contains the standard function library (which closely follows Unix C conventions), as well as the extensive Amiga library which lets you use all of Intuition's special features like window gadget control, graphics functions, sprites and the like. The only problem is that all those nifty functions are documented only in the two Intuition manuals ("Intuition – The AMIGA User Interface" and the "ROM Kernel Manual"), which are currently available only as part of the developer's kit. If you don't have the manuals and want to use the Amiga to full potential, Your best bet is to haunt the Amiga section on various Networks and BBS's in search of source code – of which there is quite a bit. Using public domain C programs as examples will give you a good idea about how to use the common Intuition library functions. If you have the set of three DOS manuals, you'll have documentation for all DOS library functions, and of course the standard Lattice C functions are documented in the C manual. That should keep you busy until you can get your hands on the Intuition manuals, which should be available from dealers soon. Even if you don't use any of Intuition's special features, programming in C is a great way to add utilities and DOS enhancements to your system.

To use Lattice C on the Amiga, you first create the source program using your favorite editor, probably the standard system editor ED for convenience sake. After saving the file with ".C" after the filename, you are ready to compile and link. To make life easy, a batch file on the C disk called "Make" calls the compiler and linker properly for you. You just EXECUTE Make from CLI, giving it the name of your source file, and wait until the work is done. If the compiling and linking was successful, you'll have a relocatable, executable object module on disk which you can run by simply typing its name, like a DOS command or any other program. Using the Icon editor from a standard WorkBench 1.1 disk, you can create a custom Icon for your program so that you can run it directly from WorkBench by just pointing and clicking.

Be forewarned: programming in C, even armed with all of the documentation you could want, is not a beginner's venture. If you're a casual weekend programmer, you may want to stick with BASIC for awhile. If, however, you're willing to learn a bit and want to produce high-quality software, then C may be the

way to go. The Amiga's system software, as well as most Amiga application software was written in C, attesting to the flexibility and speed of the language. As far as C compilers go, Lattice C is not the fastest or most efficient. The future should bring faster and possibly less expensive compilers.

68000 Assembler

The 68000 microprocessor has an instruction set which resembles that of a minicomputer like the PDP-11. It is beyond compare with any 8-bit microprocessor; if you're used to programming a 6502, the increase in power is akin to trading in your Chevette for a Turbo Porsche. Without getting into details about programming the 68000, a bit of technical data follows. The chip has eight internal data registers and eight address registers, each of which is 32 bits wide. Instructions can operate on bytes (8 bits), words (16 bits) or long words (32 bits) in memory. Addressing modes include: data or address register direct (using the value in the specified register); address register indirect (using the data in the address pointed to by the specified address register); address register indirect with post-increment (incrementing the register after an indirect); address register indirect with pre-decrement (decrementing the register before an indirect); address register indirect with displacement (register + displacement = address of data to be used); Address register indirect with index (register + displacement + another register = address of data) and others. Instructions can use any addressing mode or combination of addressing modes, for example:

```
MOVE (A4)+,100(A1,D2)
```

This single instruction means to take the data at the address pointed to by the contents of A4 (Address register 4), then increment A4; move the data to the address calculated by adding the contents of D2 (data register 2) to A1, then adding the constant 100. Machine language fans should delight in the power of the 68000.

The assembler available for the Amiga is a macro assembler with a full complement of pseudo-ops for conditional assembly, listing control, external symbol reference, etc. Also on the disk is the Linker, which is used to resolve references in your source file to external symbols. The assembler and linker work in much the same way as the C compiler and linker, but you use "Assem" and "Alink". Like C programs, your final object file will be directly executable from CLI by typing its name.

The comments in the above C section about using the ROM library routines applies in assembler also. You need the Kernel manuals to know how to use the fancy operating system functions. You also need a 512k system with two drives to use the assembler development disk. If you have such a system and the Amiga Assembler package, you can begin writing simple programs immediately. The video games may take a little longer, as you'll have to get a hold of the necessary manuals and then figure them out, which is no trivial task.

Simple Machine Language and Debugging

Up to this point, the Amiga probably sounds like a vastly powerful but somewhat remote system, with such a complex operating system that no one can directly influence the state of the machine. For those of you who like to feel firmly connected with the workings of the beast, you can actually toy with the Amiga's memory directly with the aid of a machine language monitor. A program for the Amiga called "Wack" comes with the developer's kit. If you have a copy of Wack, you can examine memory, disassemble, single step, set breakpoints, search memory, and generally have a ball. This is how the hackers find out about the machine first-hand, although on the Amiga there's a lot to find out about.

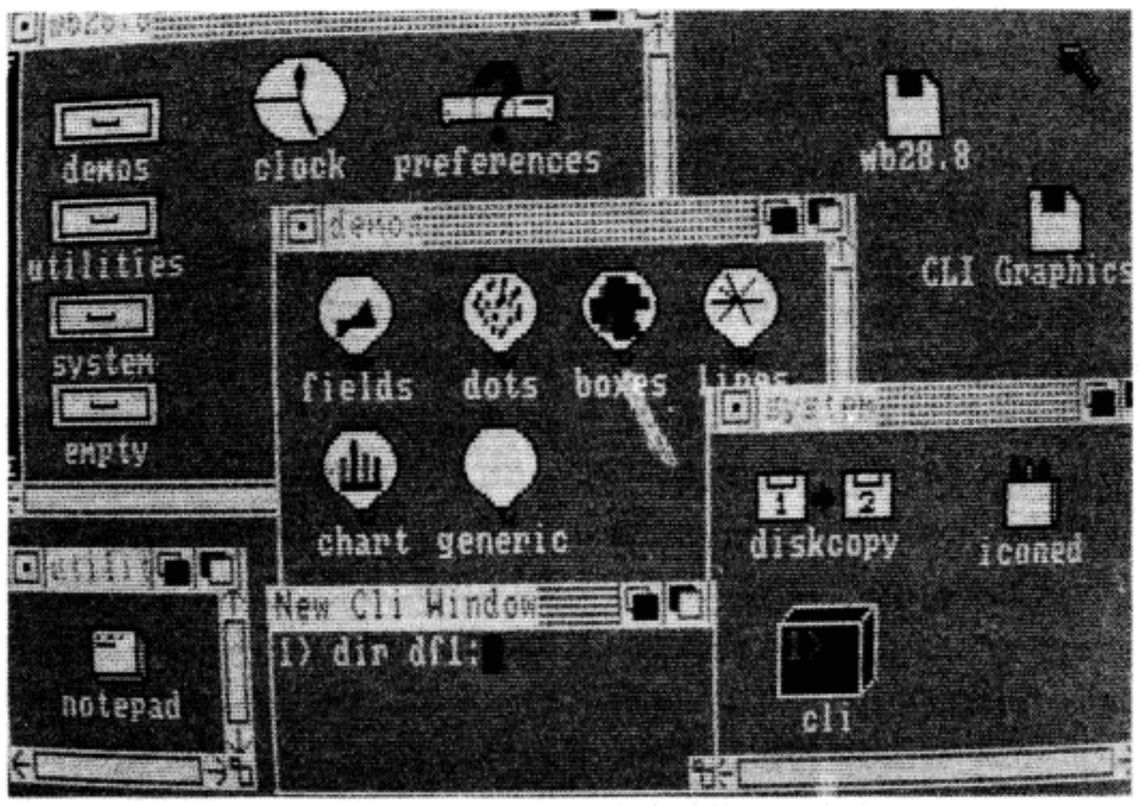
A small subset of WACK is built into the Amiga's "ROM" (actually "kickstart" RAM) and operates through a 9600 baud terminal on the serial port. You can access this "ROM Wack" directly from a WorkBench menu if you use the command "loadwb -debug" to bring up workbench instead of the usual "loadwb". Rom Wack's short command list can be displayed by typing a question-mark on the terminal. You can also enter Rom Wack after a software failure causes a "Guru Meditation #" alert to threaten bringing the system down. If you hit the RIGHT instead of the left mouse button at this point, you will be dumped into ROM Wack on the external terminal with the state of the system preserved. A handy de-bugging feature!

Other Languages

As mentioned, Pascal is also immediately available from Amiga dealers – I haven't had a chance to look at it yet. There are also a few languages in the public domain which can be had for the taking. A lot of public domain Amiga software is quite good, as it is existing C source ported from other systems. The best way to keep abreast of available software is to get a modem and snoop out any BBS that has an Amiga section or any Amiga messages. By sharing programs and ideas through user groups and networks, the Amiga knowledge and software base will spread rapidly. Things are off to a very good start already.

AMIGA DOS & CLI Commands

by Roy Reddy, Toronto, Ontario



The following article is meant as a quick reference card for those users without an AmigaDOS Manual. At time of writing Commodore states that they are supplying AMIGA developers with the AMIGA manual sets and that they hope to have these manuals available to end users in early 1986. The manuals will be sold through the same retail outlets that AMIGA computers may be purchased.

Since the manual sets are not available to every AMIGA user at this time this article becomes more timely. The mandate of this article is to offer a quick reference to the commands available in AmigaDOS. It is not possible to cover the commands in more detail than the manuals without being larger than the manuals.

The AmigaDOS environment is very helpful, in fact it has a sort of built–in help. If you remember the command but forget the pattern for the arguments you can ask AmigaDOS to show you. The syntax for this kind of help is :

< **Command** > < space > **?** < return >

Format ( ) Numeric Brackets – numeric input required
Note: < > Angle Brackets – denotes user supplied input
[ ] Square Brackets – indicates optional input

Thus: [(< >)] would indicate an optional user supplied input that must be of numeric nature if used.

AmigaDOS will respond by showing you the argument template. This template has three qualifiers which are preceded by slashes as follows:

/A – argument must be present and may NOT be omitted.
Eg. with the TYPE command, the filename is mandatory

/K – argument must use this keyword
Eg. TYPE file OPT H ;The "H" (Hex) option can only be specified by using the OPT keyword

/S – argument is used as a switch
Eg. LIST QUICK DATE ;Lists a direcory with only the filename and date – QUICK and DATE are function toggles

In future issues the command details and their applications could be further discussed but for now let us illustrate the commands that are available in AmigaDOS.

The AmigaDOS Commands Quick Reference Card

In file or directory names, a colon (:) indicates the root directory, and a slash (/) can be used to indicate a subdirectory. If any filename contains spaces, double quotes (") must enclose the entire filename. Some examples of filenames appear below.

df0:goldfish	;refers to file or directory "goldfish" in root directory of drive 0
Ark:Animals/Goat	;refers to the file or directory "goat" in the directory "Animals" on the disk "Ark"
:Bread	;File or directory bread on root directory on same disk as current directory
Foo:	;Refers to the disk named "Foo"
:Foo	;Refers to the File or directory named "Foo" in the root directory of the current-directory disk
"Music/Songs/Oh When The Saints ..."	;The file or directory "Oh When The Saints ..." in directory "Songs" in directory "Music" on current disk. Quotes are needed because of embedded spaces

File Utilities

; comment character. Allows comments for use in batch files (Executable sequences of DOS commands)

Format : [<command>] ; [<comment>]

Template : "command" ; "comment"

Examples :
copy DF0:c/list to ram: ; Copy list program to RAM drive
list DF1: ; display directory of DF1: drive

< > Direct command input and output respectively. Allows redirection of a command's input or output to a file or device

Format : <command> [<] or [>] [<arg>]

Template : "command" > "TO" < "FROM" "args"

Examples :
DATE > date-file ;send current date to "date-file"
ECHO > SER: "Message from Amiga" ;send message to serial port

COPY copies one file to another, or, copies all the files from one directory to another.

Format : COPY [FROM] <name>] [TO <name>] [ALL] [QUIET]

Template : COPY "FROM,TO/A,ALL/S,QUIET/S"

Examples :
COPY DF0: TO DF1: ALL QUIET ;copy all files from drive 0 to drive 1 without printing "copying..." messages
COPY Extras:demos/myprog TO DF1:basicdemos/ ;copy "myprog" from directory "demos" of disk "Extras" to the directory "basicdemos" of the disk in drive 1, keeping the same filename
COPY print-file TO PRT: ;send "print-file" to printer

DELETE deletes up to 10 files or directories.

Format : DELETE <name> [<name>] [ALL] [Q or QUIET]

Template : DELETE ",,,,,,,,,ALL/S,Q=QUIET/S

Examples :
DELETE out-dated-file ;delete "out-dated-file" in current directory
DELETE temps/file1 temps/file2 ;delete "file1" and "file2" in directory "temps"

DIR shows filenames in a directory.

Format : DIR [<name>] [OPT A or I or AI]

Template : DIR "DIR,OPT/K"

Examples :
DIR ;display current directory
DIR DF1: OPT A ;display entire directory structure of disk in drive 0
DIR C ;display directory "C"

ED enters a screen editor for text files.

Format : ED [FROM] <name> [SIZE <n>]
Creates a new file if <name> does not exist

Template : ED "FROM/A,SIZE"

Examples :
ED temp/ed-file
ED large-file SIZE 55000 ;Allocate up to 55,000 bytes for file

EDIT enters a line by line editor.

Format : EDIT [FROM] <name> [TO] <name>]
[WITH <name>] [VER <name>] [OPT <option>]

Template : EDIT "FROM/A,TO,WITH/K,VER/K,OPT/K"

Examples :
EDIT ed-file WITH edits VER nil: ;Get edit commands from "edits", edit "ed-file", and do not print any verification of edit commands
EDIT orig-file TO new-file

FILENOTE attaches a note with a maximum of 80 characters to a specified file.

Format : FILENOTE [FILE] <file> COMMENT <string>

Template : FILENOTE "FILE/A,COMMENT/K"

Examples :
FILENOTE my-picture COMMENT "drawn in November"
FILENOTE src-file COMMENT "source for screen"

JOIN concatenates up to 15 files to form a new file.

Format : JOIN <name> <name> [<name>] AS <name>

Template : JOIN ",,,,,,,,,,,,,,,AS/A/K"

Examples : JOIN src-file1 src-file2 AS all-src
JOIN text data results AS experiment

LIST examines and displays detailed information about a file or directory.

Format : LIST [DIR] <dir> [P or PAT <pat>] [KEYS] [DATES] [NODATES] [TO <name>] [S <string>] [SINCE <date>] [UPTO <date>] [QUICK]

Template : LIST "DIR,P=PAT/K,KEYS/S,DATES/S, NODATES/S,TO/K,S/K,SINCE/K,UPTO/K,QUICK/S"

Examples :
LIST DF1: ;List all files on disks in root directory of disk in drive 1
LIST P ?? ;List all files in current directory with names two characters in length
LIST :l S handler ;List all files in directory "l" in root directory with the characters "handler" somewhere in the filename
LIST SINCE YESTERDAY ;List all files created since yesterday

MAKEDIR creates a directory with a specified name.

Format : MAKEDIR <dir> Template : MAKEDIR "/A"

Examples :
MAKEDIR DF0:test ;Creates directory "test" in root directory of disk in drive 0
MAKEDIR RAM:temp/files ;make directory "files" in directory "temp" in root directory of RAM-Disk

PROTECT sets a file's protection status.

Format : PROTECT [FILE] <name> [FLAGS <status>]

Template : PROTECT "FILE,FLAGS/K"

Examples :
PROTECT file1 rwd ;Allow "file1" for Read, Write and Deletion
PROTECT temp/file FLAGS r ;Only allow "temp/file" to be Read

RENAME renames a file or directory.

Format : RENAME [FROM] <name> [TO or AS] <name>

Template : RENAME "FROM/A,TO=AS/A"

Examples :
RENAME OLD-NAME NEW-NAME
RENAME ram:c/file1 AS new-file ;

SEARCH looks for a specified text string in all the files of a directory.

Format : SEARCH [FROM] <name> or <pattern> [SEARCH] <string> [ALL]

Template : SEARCH "FROM,SEARCH/A,ALL/S"

Examples :
SEARCH DF0: my-name ;Search for the text "my-name" in the root directory of the disk in drive 0
SEARCH DF1: source ALL ;Search for the text in the root directory and all the directories therein on drive 1

SORT sorts simple files.

Format : SORT [FROM] <name> [[TO] <name>] [COLSTART <n>]

Template : SORT "FROM/A,TO/A,COLSTART/K"

Examples :
SORT file TO sorted-file
SORT list TO sort-list COLSTART 5 ;Sort using data starting at fifth byte in each record as sort key

TYPE types a file to the screen that you can optionally specify as text or hex.

Format : TYPE [FROM] <name> [[TO <name>] [OPT N or H]

Template : TYPE "FROM/A,TO,OPT/K"

Examples :
TYPE preferences OPT H ;print hex dump of "preferences"
TYPE :s/startup-sequence ;print file "startup-sequence" in "s" dir.
TYPE some-file TO ser: ;send "some-file" to the serial port

CLI Control

BREAK sets attention flags in a given process.

Format : BREAK <task> [ALL] [C] [D] [E] [F]

Template : BREAK "TASK/A,ALL/S,C/S,D/S,E/S,F/S"

Examples :
BREAK 6 ;Send a CTRL-C to task # 6
BREAK 4 D F ;Sends a CTRL-D and CTRL-F to task 4

CD sets a current directory and/or drive.

Format : CD [<dir>] Template : CD "DIR"

Examples :
CD RAM: ;Set root directory in RAM-disk as the current directory
CD DF1:temp/l ;set directory "l" in directory "temp" on disk in drive 1 as the current directory

ENDCLI ends an interactive CLI process.

Format : ENDCLI Template : ENDCLI

Examples : ENDCLI

NEWCLI creates a new interactive CLI process.

Format : NEWCLI [<window>]

Template : NEWCLI "WINDOW"

Examples :
NEWCLI
NEWCLI CON:30/30/300/120/"NEWEST CLI" ;Create a new CLI process in a window titled "NEWEST CLI" starting at screen co-ordinates (30,30), 300 pixels wide and 120 pixels high

PROMPT changes the prompt in the current CLI.

Format : PROMPT <prompt>

Template : PROMPT "PROMPT"

Examples :
PROMPT
PROMPT "%n>" ;Put current CLI process number in prompt

RUN executes commands as a background process. Continues CLI as command executes

Format : RUN <command>

Template : RUN command +
command

Examples :
RUN TYPE :s/startup-sequence
RUN COPY :c/list TO RAM: +
CD RAM: +
LIST

STACK displays or sets the stack size for commands.

Format : STACK [<n>]

Template : STACK "SIZE"

Examples :
STACK
STACK 9000

STATUS displays information about the CLI processes currently in existence.

Format : STATUS [<process>] [FULL] [TCB] [SEGS] [CLI or ALL]

Template : STATUS "PROCESS,FULL/S,TCB/S,SEGS/S,
CLI=ALL/S"

Examples :
STATUS
STATUS 1 FULL

WHY explains why a previous command failed.

Format : WHY

Template : WHY

Examples :
WHY

Command Sequence Control

ECHO displays the message specified in a command arg.

Format : ECHO < string > Template : ECHO " "

Examples : ECHO "This string was echoed to the screen"

EXECUTE executes a file of commands.

Format : EXECUTE <commandfile> [arguments]

Template : EXECUTE "command-file","args"

Examples : EXECUTE :s/startup-sequence

FAILAT fails a command sequence if a program returns an error code greater than or equal to specified number.

Format : FAILAT <n> Template : FAILAT "rclim"

Examples : FAILAT
FAILAT 14

IF tests specified actions within a command sequence.

Format : IF [NOT] [WARN] [ERROR] [FAIL] [<string> EQ <string>] [EXISTS <name>]

Template : IF "NOT/S,WARN/S,ERROR/S,FAIL/S,,EQ/K,EXISTS/K"

Examples :
IF EXISTS :c/cd ;execute commands up to ENDIF if the file "cd" is in directory "c" on root directory

LAB defines a label (see SKIP).

Format : LAB <string> Template : LAB <text>

Examples : LAB error-location
LAB ok

QUIT exits from command sequence with a given error code.

Format : QUIT [<returncode>] Template : QUIT "RC"

Examples : QUIT 14
QUIT

SKIP jumps forward to a LAB in a command sequence (see LAB).

Format : SKIP <label> Template : SKIP "LABEL"

Examples : SKIP
SKIP error-location

WAIT waits for, or until, a specified time.

Format : WAIT <n> [SEC or SECS] [MIN or MINS] [UNTIL<time>]

Template : WAIT ",SEC=SECS/S,MIN=MINS/S,UNTIL/K"

Examples : WAIT ;Wait one second
WAIT UNTIL 12:25
WAIT 5 MIN

System and Storage Management

ASSIGN assigns a logical device name to a directory or filename.

Format : ASSIGN [[<name>] <dir>] [LIST]

Template : ASSIGN "NAME,DIR,LIST/S"

Examples :
ASSIGN temp: DF0:source/files
ASSIGN ;Show all current assignments

DATE displays or sets the system date and time.

Format : DATE [<date>] [<time>] [TO or VER <name>]

Template : DATE "DATE,TIME,TO=VER/K"

Examples :
DATE ;Display today's date
DATE TOMORROW ;Advance the current date by one day
DATE 01-OCT-85 12:32
DATE TUESDAY ;Set the date to the upcoming Tuesday

DISKCOPY copies the contents of one entire microdisk to another.

Format : DISKCOPY [FROM] <disk> TO <disk> [NAME <name>]

Template : DISKCOPY "FROM/A,TO/A/K,NAME/K"

Examples :
DISKCOPY FROM DF0: TO DF1:
DISKCOPY DF0: TO DF0: NAME "Copied Disk"

FAULT displays messages corresponding to supplied fault or error codes.

Format : FAULT [<n>] Template : FAULT ",,,,,,,,,"

Examples :
FAULT 123 ;Display error message #123
FAULT 133 234 245 ;Display list of messages for 133, 234 and 245

FORMAT formats and initializes a new 3 1/2 inch floppy disk.

Format : FORMAT DRIVE <drivename> NAME <string>

Template : FORMAT "DRIVE/A/K,NAME/A/K"

Examples : FORMAT DRIVE DF0: NAME "New Blank Disk"

INFO gives information about the filing system, including room left on all mounted volumes.

Format : INFO Template : INFO Examples : INFO

INSTALL makes a formatted disk bootable.

Format : INSTALL [DRIVE] <drive>

Template : INSTALL "DRIVE/A" Examples : INSTALL DF0:

RELABEL changes the volume name of a disk.

Format : RELABEL [DRIVE] <drive> [NAME] <name>

Template : RELABEL "DRIVE/A,NAME/A"

Examples : RELABEL DF1: "Disk over there"

AMIGA Editor Commands

by Roy Reddy, Toronto, Ontario

There are two editors that come with every AMIGA as part of the WORKBENCH diskette. The names are 'ED' and 'EDIT'. They are, respectively, a screen editor and a line editor. This article is meant to be a quick reference to the commands available in each editor.

ED – The Screen Oriented Editor

The template for the screen editor "ED" is :

ED "FROM/A, SIZE/K"

. . .where the 'FROM' argument is the file to be edited. ED allows for a file of 40,000 bytes as a default if more memory is required then the 'SIZE' switch may be used to allocate more memory.

'ED' has two categories of commands: immediate commands and extended commands. Immediate commands are a sequence of keystrokes performed while in the middle of a document. Some immediate commands are a combination of the CTRL key and another key. Extended commands are executed in a command mode. The 25th row of the screen is reserved for the extended command line and is entered by striking the ESC key. Some commands are available in both modes. The following list will comprise the quick reference for 'ED'.

Immediate Commands

Cursor Movement Immediate Commands

Cursor UP	moves cursor UP	**DEL**	deletes character under cursor and moves remaining text left.
Cursor DOWN	moves cursor DOWN	**ESC**	Enters the extended command mode at bottom of screen.
Cursor RIGHT	moves cursor RIGHT	**RETURN**	return cursor to left edge and if in insert mode the current line will be broken in two.
Cursor LEFT	moves cursor LEFT		
BACKSPACE	moves cursor left and erases char.	**TAB**	moves cursor to next TAB position right.

CTRL Key Commands

CTRL+A	Insert a Line after the current line	**CTRL+I**	equivalent to TAB
CTRL+B	Delete the current line	**CTRL+M**	equivalent to RETURN
CTRL+D	almost equivalent to CURSOR UP, scrolls text down	**CTRL+O**	deletes spaces until next non–space character or deletes character as DEL if not a space
CTRL+E	move to opposite (top or bottom) corner of screen.	**CTRL+R**	moves cursor to the space following the previous word
CTRL+F	flips the case of the character under the cursor	**CTRL+T**	moves cursor to the start of the next word (non–space)
CTRL+G	will repeat the last Extended command	**CTRL+U**	almost equivalent to CURSOR DOWN, scrolls text up
CTRL+H	equivalent to BACKSPACE	**CTRL+V**	will update or redraw the screen
		CTRL+[	equivalent to ESC
		CTRL+]	move cursor to opposite (left or right) end of current line

Extended Commands

ESC or CTRL+[starts the extended mode. Some extended commands can contain additional parameters (string or numeric). Strings must be delimited by a character other than letters, space, numbers, brackets, or a semicolon. The slash character "/" is an acceptable delimiter and will be used in the examples in this article.

A /str/	insert a line of text "str" After current line	**J**	Joins the current line with the next
B	move cursor to the Bottom of the file	**LC**	set ED to distinguish between upper/Lower Case
BE	set the Block End at cursor position	**M n**	Moves the cursor to the line specified by n
BF /str/	search for 'string' Backwards	**N**	moves the cursor to the start of the Next line
BS	set the Block Start at cursor position	**P**	moves the cursor to the start of the Previous line
CE	move the Cursor to the End of the line	**RP**	Repeat the last extended command continuously
CL	equivalent to Cursor Left in immediate mode	**S**	create a new line by Splitting the current line
CR	equivalent to Cursor Right in immediate mode	**SA**	Save the document
CS	move the Cursor to the Start of the line	**SB**	moves cursor to the Start of the defined Block
D	Delete the current line	**SH**	Show the status (margins, tab length, filename)
DB	Delete the previously defined Block	**SL n**	Set the Left margin to the value specified in n
DC	equivalent to DEL in immediate mode	**SR n**	Set the Right margin to the value specified in n
E /str1/str2/	Exchange "str2" for "str1" throughout document	**ST n**	Set the distance between each TAB to the value specified in n
EQ /str1/str2/	as above but will Query before Exchange		
EX	Extend right margin ignoring defaults	**T**	moves the cursor to the Top of the document
F /str/	Find "str1" in document	**U**	Undo the last change made (except delete line)
I /str/	Insert a line of text "str" before current line	**UC**	set ED to NOT distinguish between Upper/lower Case
IB	Insert a copy of a previously defined Block	**WB /fil/**	Write previously defined Block to file "fil"
IF /fil/	Insert or merge the File "fil" into the document at the current line	**X**	eXit from ED & write document to file

EDIT: Amiga's Line Oriented Editor

by Roy Reddy, Toronto, Ontario

This article describes the line oriented Editor of AmigaDOS called 'EDIT'. The documentation covering 'EDIT' in the Amiga manual "AmigaDOS User's Manual" is 35 pages long. This article and the following pages will act as a quick reference guide only and cannot cover this Editor in as much detail as the manual.

Commodore full screen Editors have spoiled me thus making it hard to get interested in studying a line Editor. EDIT however offers quite a bit of power in that it can modify files using commands from another file called an "Edit Command File". The following describes EDIT's format and template :

Format : EDIT [FROM]<name> [TO]<name>] [WITH<name>]
[VER<name>] [OPT Pn:Wn:PnWn]

Template : EDIT " FROM/A,TO,WITH/K,VER/K,OPT/K "

EDIT has some limitations, the first of which, you will discover, is that you cannot create a new file using EDIT. For this reason the 'FROM' argument must exist even though the keyword 'FROM' is optional.

The argument 'TO' instructs EDIT what filename to give the destination file. If the 'TO' argument is omitted EDIT creates a file and will rename it with the 'FROM' name when you Quit EDIT. The original file is saved as ':T/EDIT-BACKUP' and will remain until the next EDIT session.

The 'WITH' file is the "EDIT Command File" which gives EDIT its additional editing power. The 'VER' file, if specified, will contain error messages that may have been generated during the EDIT session. If either the 'WITH' or 'VER' arguments is omitted, EDIT will use the keyboard and screen for input and output respectively.

EDIT has memory and line width defaults that can be adjusted using the 'OPT' keyword and either/both the 'P' & 'W' arguments. With the 'P' argument you can adjust the amount of memory EDIT uses to retain previous lines. The 'W' argument adjusts the maximum line length used by EDIT. The default setting of these parameters is P40W120 ; 40 previous lines retained & 120 maximum line length.

This table describes the abbreviations used in this quick reference guide.

/qs/	qualified string	**<f>**	file specifier
/t/	string	**sw**	+ or – (on or off)
n	line number, or.		

Character Pointer Commands

<	move character pointer left	**$**	lower case character at pointer	**PA /qs/**	move character pointer to After /qs/
>	move character pointer right	**%**	upper case character at pointer	**PB /qs/**	move character pointer to Before /qs/
#	delete character at pointer	**<SPACE>**	turn character at pointer to space	**PR**	reset character pointer to start of line

Current Line Positioning Commands

M n	move to line n	**N**	move to Next line in memory	**F /qs/**	search for string /qs/
M +	move to highest line in memory	**P**	move to Previous line in memory	**BF /qs/**	search Backward for string /qs/
M –	move to lowest line in memory	**REWIND**	make current line line #1 of source file	**DF /qs/**	search & Delete line with string /qs/

Text Display and Verification Commands

?	verify current line
!	verify with case indicators
T	type lines until end of file
T n	type n lines forward
TL n	type n lines with line numbers
TP	moves to lowest line then type lines
V sw	switch to turn on/off line display verification

Commands That Operate Globally

GA /qs/t/	Globally place string /t/ After string /qs/
GAB /qs/t/	Globally place string /t/ Before string /qs/
GE /qs/t/	Globally exchange string /t/ for string /qs/
CG [n]	Cancel Global operation [n] (all operations if [n] omitted)
DG [n]	Disable Global operation [n] (all operations if [n] omitted)
EG [n]	Enable Global operation [n] (all operations if [n] omitted)
SHG	Show information about Globals in use

Commands That Operate On The Current Line

A /qs/t/	insert string /t/ After string /qs/ on current line
AP /qs/t/	same as above but moves character pointer
B /qs/t/	insert string /t/ Before string /qs/ on current line
BP /qs/t/	same as above but moves character pointer
CL t	Concatenate current line + string /t/ + next line
D	Delete current line
DFA /qs/	Delete After string /qs/ to the end of the line
DFB /qs/	Delete Before string /qs/ to the end of the line
DTA /qs/	Delete from start of line to After string /qs/
DTB /qs/	Delete from start of line to Before string /qs/
E /qs/t/	Exchange string /qs/ for string /t/
EP /qs/t/	same as above but moves character pointer
I	Insert chars from keyboard before chars in line
I <f>	Insert chars from file <f> before chars in line
R	Replace characters from keyboard
R <f>	Replace characters from file <f>
SA /qs/	Split current line After string /qs/
SB /qs/	Split current line Before string /qs/

Input/Output Redirection

FROM	read From source (original file)
FROM <f>	read From file <f>
TO	return to original destination file
TO <f>	send output to file <f>
CF <f>	Close file <f>

Miscellaneous Commands

,	repeat previous A, B, or E command
= n	set line number to n
C <f>	take Commands from file <f>
H n	set Halt at line n (if n = * then Halt and unset H)
Q	Quit input from command file, or windup if no command file
SHD	Show Data ; last cmd, search string
STOP	Stop ; quit without changes made to existing file
TR sw	switch to suppress Trailing spaces from lines
W	Windup ; continue through the remaining source file
Z /t/	change value of current input terminator string to /t/

Pick Areas and Pop Menus

Darren Spruyt
Gravenhurst, Ontario

Drop–Down Menus For Your Commodore 64

Such a title definitely needs explaining. While reading other computer literature, you may have heard the term 'POP MENUS'. 'Pick Areas' are just as common, but less documented. These features are now available on your C–64 with the help of a short assembly language program.

Pop Menus are just what they sound like. A menu that can be 'popped on' the screen for a decision to be made, then be 'popped off' again when the decision has been made. You can do just about the same in BASIC, however the machine language version presented hear has more features. It automatically remembers what was present on the screen when it was 'popped on' and will restore the information back to the screen again when it is 'popped off,' as well as remembering the correct colors. The pop menu on the screen also comes complete with a border and a reverse field line for selecting the appropriate item in the menu.

Pick areas are areas on the screen given a (X,Y) position, width and height and finally an ID number. When a pointer on the screen is moved into a Pick Area, the area is highlighted (reversed) and a flag is set to tell the main program that the pointer has been moved into a Pick Area. This may not seem like much, but is paves the way for powerful menu and selecting features to be added to a basic program very easily. The pointer is an integral part of this package and it operates from the cursor keys when they are not in use. (i.e. during a run of a BASIC program when no input is being accepted).

The PICK AREA and POP MENU machine language program is very easy to use. There are just a few SYS's to use (5) and you can be on your way to a great looking program. The first SYS turns on the PICK program, it is:

```
SYS 49152
```

This turns on the sprite to be used as the pointer and sets up the required information for the program to operate. The second SYS defines a Pick Area:

```
SYS 49155,X,Y,W,H
```

Given the screen as a grid of 0–39 horizontally and 0–24 vertically from the top left corner, (X,Y) is the top left corner of the Pick Area with width of W and height of H. The ID number of the Pick Area is in location 782 (values from 1–16 are normal) so retrieve it with a PEEK and remember it for further use.

```
SYS 49161,X
```

. . .will delete the Pick Area with ID X from the list of active Pick Areas.

```
SYS 49164
```

. . .will turn off the PICK program and remove the pointer from the screen. While the PICK program is active, if the pointer enters a PICK area, it's ID will be placed in memory location 2.

The last SYS is for use with the POP MENUs, and the PICK program does not have to be active for the POP MENUs to work. To use a pop–menu,

```
SYS 49158,X,Y,W,H,C,A$
```

. . . will create a pop menu at (X,Y) as mentioned earlier with width W and height H using color C and using the last parameter as the string of data with which to fill the menu. Since the pop–menu places a border around it, the width for text inside the menu will be (H–2) and the number of lines allowable will be (V–2). The string needs no cursor characters to be included within it – the text will 'wrap' inside the area automatically.

If a menu was to be created having 4 lines of text with each line being 6 characters long, and to hold the following information on separate lines 'WHITE,BLACK,YELLOW,ORANGE' the following would be done. Define a string variable such as:

```
A$ = "WHITE  BLACK  YELLOWORANGE"
SYS 49158,0,0,8,6,1,A$
```

This would place a pop menu at (0,0) with the information within it. NOTE: the string is to be exactly as written above, the spaces or lack thereof is important. Once this has been done, the menu will be on the screen and a reverse bar will be on the first line of the menu. This line is moved up and down with the cursor keys to select the item and then the return is pressed. This will then remove the pop–menu from the screen, replacing the old data, while a number corresponding to which line the selector bar was on will be placed in memory location 599. The contents of 599 would then be used to index to the chosen operation, which in this case would be to the correct colour value.

It may seem complex, but is really very simple to use, also very powerful. With such a utility, Basic can be used to produce very graphically appealing and easy to use programs. A sample program using Pick Areas and two pop–menus appears in Listing 1, while Listing 2 is a program to create the machine language file on the disk.

Pop Menus Demo Program

```
AN  95 if fl = -1 then 200
JF  100 a = peek(49152) + peek(49153)*256 + peek(49154)
AH  110 if a = 4108 then 200
MB  120 fl = -1 :load " pick.mlp.c000 " ,8,1
MD  200 :
LL  300 h1$ = chr$(192) + chr$(192):h2$ = h1$ + h1$
OA  310 v$ = chr$(221)
CL  320 ul$ = chr$(176) : rem upper left corner (c = & a)
KN  330 ur$ = chr$(174) : rem upper right corner (c = & s)
OM  340 ll$ = chr$(173) : rem lower left corner (c = & z)
IP  350 lr$ = chr$(189) : rem lower right corner (c = & x)
HF  1000 rem window using program
OG  1010 print " S " ;
ND  1030 printul$h2$chr$(178)h2$h2$h2$h1$chr$(178)
        h2$h2$h2$h2$h1$ur$;
CO  1040 print v$ " exit " v$ " [14 spcs] " v$ " background
        [8 spcs] " v$;
EB  1050 printchr$(171)h2$chr$(219)h2$h2$h2$h1$
        chr$(177)h2$h2$h2$h2$h1$lr$;
MN  1060 printv$ " time " v$
CM  1070 printv$ " on[2 spcs] " v$
FK  1080 print chr$(171)h2$chr$(179)
PC  1090 printv$ " opts " v$
IM  1100 printll$h2$lr$
ME  1200 gosub 60000
JI  1210 geta$:
II  1212 print " sq]]]]]] " ;b:if z = 0 then b = b + 1
HM  1214 if a$<>chr$(13) then 1210
KI  1220 if peek(2) = ex then 20000
PH  1230 if peek(2) = bk then 21000
FO  1250 if peek(2) = tm then 23000
KG  1900 goto 1210
GD  20000 a$ = " no yes "
GF  20010 sys 49158,0,0,5,4,1,a$
CB  20020 if peek(599) =2 then sys 49164:stop
MD  20030 goto 1210
MJ  21000 a$ = " black white red[3 spcs]cyan[2 spcs]
        purplegreen blue[2 spcs]yellow "
OM  21010 sys 49158,21,0,8,10,0,a$
PH  21020 poke 53281,peek(599)-1
EC  21030 goto 1210
FM  23000 if z = 0 then 23100
MG  23010 z = 0
DE  23020 print " sqqq]r time q [4 lefts]on[2 spcs] "
EP  23030 goto1210
KF  23100 rem
EN  23110 z = 1
HN  23120 print " sqqq]r time q [4 lefts]off[1 spc] "
IF  23130 goto1210
NE  60000 sys 49152
FC  60010 sys 49155,1,1,4,1:ex = peek(782)
BD  60020 sys 49155,1,3,4,2:tm = peek(782)
CE  60030 sys 49155,1,6,4,1:op = peek(782)
AB  60040 sys 49155,21,1,10,1:bk = peek(782)
OK  60050 return
```

Pop Menus BASIC Loader

```
CC  100 open 15,8,15
IN  110 print#15, " s0:pick.mlp.c000 "
PP  120 open 1,8,2, " pick.mlp.c000,p,w "
JE  130 print#1,chr$(0);chr$(192);
JH  140 rem start of basic loader code
NH  150 read a,b,d
IL  160 print " now loading in code. "
EK  170 for k = a to b
MF  180 read c:print#1,chr$(c);:
NH  190 poke 1024,c:poke55296,c
IG  200 ch = ch + c:next:close1
AL  210 if ch<>d then print " data error " :stop
MA  220 print " done. " :end
PD  999 data  49152, 50321, 134294
KF  1000 data  76, 164, 195,  76, 245, 193,  76,  16
HN  1010 data 192,  76, 185, 193,  76, 201, 193,  96
HC  1020 data 160,   0,  32,  11, 194,  48, 248, 173
IP  1030 data 129, 196, 201,   3, 144, 241, 173, 112
JN  1040 data 196, 201,   3, 144, 234, 140,  88,   2
FK  1050 data  32, 234, 195, 142, 149, 196,  32, 253
GF  1060 data 174,  32, 158, 173, 164, 101, 165, 100
AM  1070 data  32, 219, 182, 160,   2, 177, 100, 133
MO  1080 data  72, 136, 177, 100, 133,  71, 165,  71
AH  1090 data 208,   2, 198,  72, 198,  71, 169,   0
GF  1100 data  32,  36, 193, 160,   0,  32, 127, 195
DH  1110 data 174, 129, 196,  32,  20, 193, 172, 112
PH  1120 data 196, 136, 169, 110, 145, 251, 136,  48
HH  1130 data  13, 169,  64, 145, 251, 136,  48,   6
NJ  1140 data 208, 249, 169, 112, 145, 251, 142, 146
ML  1150 data 196, 169,  40, 162, 251,  32, 222, 193
IH  1160 data  32,  20, 193, 174, 146, 196, 202, 240
EM  1170 data  72, 224,   1, 240,  44, 160,   0, 169
AM  1180 data  93, 145, 251, 200, 204, 112, 196, 240
FB  1190 data  10, 177,  71,  32, 232, 193, 145, 251
LE  1200 data  76, 147, 192, 136, 169,  93, 145, 251
PO  1210 data 142, 146, 196, 173, 112, 196,  56, 233
FL  1220 data   2, 162,  71,  32, 222, 193,  76, 121
NB  1230 data 192, 172, 112, 196, 136, 169, 125, 145
DF  1240 data 251, 169,  64, 136, 240,   7,  48,   9
KG  1250 data 145, 251,  76, 195, 192, 169, 109, 145
GE  1260 data 251, 162,   1,  32, 153, 193, 142, 146
FG  1270 data 196,  32, 228, 255, 174, 146, 196, 201
KA  1280 data 145, 208,  11, 224,   1, 240, 239,  32
AG  1290 data 153, 193, 202,  76, 211, 192, 201,  17
PC  1300 data 208,  16, 138,  24, 105,   2, 205, 129
OI  1310 data 196, 240, 219,  32, 153, 193, 232,  76
DI  1320 data 211, 192, 201,  13, 208, 208, 142,  87
JG  1330 data   2, 169, 128,  32,  36, 193, 169,   1
MH  1340 data 141,  88,   2,  96, 172, 112, 196,  32
CK  1350 data 130, 193, 136, 173, 149, 196, 145, 253
IJ  1360 data 136,  16, 251,  96, 141, 147, 196, 169
PE  1370 data   0, 133,  34, 169, 176, 133,  35, 160
DF  1380 data   0,  32, 127, 195, 165,   1,  41, 254
AL  1390 data 120, 133,   1,  32, 130, 193, 174, 129
IO  1400 data 196, 172, 112, 196, 136, 173, 147, 196
PI  1410 data  16,  11, 177,  34, 145, 251, 177,  36
CN  1420 data 145, 253,  76,  93, 193, 177, 251, 145
HG  1430 data  34, 177, 253, 145,  36, 136,  16, 229
EP  1440 data 142, 146, 196, 169,  40, 162, 251,  32
BE  1450 data 222, 193, 169,  40, 162,  34,  32, 222
HA  1460 data 193,  32, 130, 193, 174, 146, 196, 202
```

```
CN 1470 data 208, 199, 165,   1,   9,   1, 133,   1
BP 1480 data  88,  96, 165,  34, 133,  36, 165,  35
DA 1490 data   9,   4, 133,  37, 165, 251, 133, 253
FB 1500 data 165, 252,  41,   3,   9, 216, 133, 254
JA 1510 data  96, 160,   0, 138, 141, 146, 196,  24
DD 1520 data 109,  95, 196, 170,  32, 130, 195, 172
GC 1530 data 112, 196, 136, 136, 177, 251,  73, 128
OE 1540 data 145, 251, 136, 208, 247, 174, 146, 196
NA 1550 data  96,  32, 234, 195, 224,  17, 176,   6
AG 1560 data 169,   0, 157,  61, 196,  96, 169, 255
AE 1570 data  96, 173,  21, 208,  41, 127, 141,  21
NK 1580 data 208, 120, 169, 234, 141,  21,   3, 169
OP 1590 data  49, 141,  20,   3,  88,  96,  24, 117
PC 1600 data   0, 149,   0, 144,   2, 246,   1,  96
KG 1610 data 201,  64, 144,   2, 233,  64, 201, 128
MG 1620 data 144,   2, 233,  64,  96, 160,  16, 185
CB 1630 data  61, 196, 240,   6, 136, 208, 248, 169
LK 1640 data 254,  96,  32,  11, 194,  48, 250, 153
KK 1650 data  61, 196,  96,  32, 234, 195, 201,  40
EE 1660 data 176,  48, 153,  78, 196,  32, 234, 195
AM 1670 data 201,  25, 176,  38, 153,  95, 196,  32
OO 1680 data 234, 195, 240,  30, 153, 112, 196,  24
IO 1690 data 121,  78, 196, 201,  40, 176,  19,  32
LC 1700 data 234, 195, 240,  14, 153, 129, 196,  24
LA 1710 data 109,  95, 196, 201,  25, 176,   3, 169
DO 1720 data   1,  96, 169, 255,  96, 169, 194,  72
CC 1730 data 169,  82,  72,   8,  72,  72,  72,  76
KJ 1740 data  49, 234, 173,  88,   2, 208,   3,  76
DE 1750 data 188, 254, 165, 157,  48, 249, 165, 204
FC 1760 data 240, 245, 164, 198, 185, 118,   2, 162
OO 1770 data   3, 221,  55, 196, 240,   5, 202,  16
GI 1780 data 248,  48,   2, 198, 198, 165, 203, 201
KE 1790 data   7, 240,   4, 201,   2, 208, 216,  41
BA 1800 data   1, 172, 141,   2, 240,   9, 192,   3
ND 1810 data 176, 205,   9,   2,  76, 143, 194, 201
AC 1820 data   3, 208,  12, 172,  59, 196, 240,  56
CI 1830 data 136, 140,  59, 196,  76, 208, 194, 201
NF 1840 data   1, 208,  14, 172,  59, 196, 192,  99
FP 1850 data 176,  38, 200, 140,  59, 196,  76, 208
FG 1860 data 194, 201,   2, 208,  12, 172,  60, 196
FO 1870 data 240,  22, 136, 140,  60, 196,  76, 208
BH 1880 data 194, 201,   0, 208,  11, 172,  60, 196
IN 1890 data 192, 159, 176,   4, 200, 140,  60, 196
NP 1900 data 173,  16, 208,  41, 127, 141,  16, 208
BI 1910 data 173,  60, 196,  10, 144,   3,  32, 153
II 1920 data 195,  24, 105,  24, 144,   3,  32, 153
JF 1930 data 195, 141,  14, 208, 173,  59, 196,  10
JP 1940 data 105,  50, 141,  15, 208, 173,  60, 196
AA 1950 data  74,  74, 141,  86,   2, 173,  59, 196
DO 1960 data  74,  74, 141,  85,   2, 160,  16, 185
HN 1970 data  61, 196, 240,  55, 173,  86,   2, 217
PK 1980 data  78, 196, 144,  47, 249,  78, 196, 217
BD 1990 data 112, 196, 176,  39, 173,  85,   2, 217
FK 2000 data  95, 196, 144,  31, 249,  95, 196, 217
PC 2010 data 129, 196, 176,  23, 196,   2, 240,  16
LD 2020 data 140, 148, 196, 164,   2,  32,  82, 195
FE 2030 data 172, 148, 196, 132,   2,  32,  82, 195
EC 2040 data  76, 188, 254, 136, 208, 193, 164,   2
AJ 2050 data  32,  82, 195, 169,   0, 133,   2,  76
OE 2060 data 188, 254, 152, 240,  41,  32, 127, 195
HH 2070 data 190, 129, 196, 185, 112, 196, 141, 146
KJ 2080 data 196, 206, 146, 196, 172, 146, 196, 177
HK 2090 data 251,  73, 128, 145, 251, 136,  16, 247
FE 2100 data 165, 251,  24, 105,  40, 133, 251, 144
EG 2110 data   2, 230, 252, 202, 208, 230,  96, 190
HJ 2120 data  95, 196, 181, 217,  41,   3,  13, 136
GJ 2130 data   2, 133, 252, 189, 240, 236,  24, 121
MC 2140 data  78, 196, 133, 251, 144,   2, 230, 252
KJ 2150 data  96,  72, 173,  16, 208,   9, 128, 141
CB 2160 data  16, 208, 104,  96, 160,  63, 185, 248
OA 2170 data 195, 153, 192,   3, 136,  16, 247, 120
CN 2180 data 169, 194, 141,  21,   3, 169,  69, 141
NJ 2190 data  20,   3,  88, 169, 128, 141,  21, 208
EP 2200 data 169,  15, 141, 255,   7, 141,  88,   2
AG 2210 data 169,   0, 141,  61, 196, 133,   2, 141
EB 2220 data  60, 196, 141,  59, 196, 169,   0, 160
HE 2230 data  16, 153,  61, 196, 136,  16, 250, 169
GM 2240 data  24, 141,  14, 208, 169,  50, 141,  15
OC 2250 data 208,  96, 140, 146, 196,  32, 253, 174
PJ 2260 data  32, 158, 183, 172, 146, 196, 138,  96
CN 2270 data 254,   0,   0, 224,   0,   0, 240,   0
NM 2280 data   0, 216,   0,   0, 204,   0,   0, 198
KM 2290 data   0,   0,   3,   0,   0,   1,   0,   0
KM 2300 data   0,   0,   0,   0,   0,   0,   0,   0
EN 2310 data   0,   0,   0,   0,   0,   0,   0,   0
ON 2320 data   0,   0,   0,   0,   0,   0,   0,   0
IO 2330 data   0,   0,   0,   0,   0,   0,   0,   0
BG 2340 data   0,   0,   0,   0,   0,   0,   0,  17
BO 2350 data  29, 145, 157,   0,   0,   0,   0,   0
GA 2360 data   0,   0,   0,   0,   0,   0,   0,   0
AB 2370 data   0,   0,   0,   0,   0,   0,   0,   0
KB 2380 data   0,   0,   0,   0,   0,   0,   0,   0
EC 2390 data   0,   0,   0,   0,   0,   0,   0,   0
OC 2400 data   0,   0,   0,   0,   0,   0,   0,   0
ID 2410 data   0,   0,   0,   0,   0,   0,   0,   0
CE 2420 data   0,   0,   0,   0,   0,   0,   0,   0
ME 2430 data   0,   0,   0,   0,   0,   0,   0,   0
GF 2440 data   0,   0,   0,   0,   0,   0,   0,   0
AG 2450 data   0,   0,   0,   0,   0,   0,   0,   0
CB 2460 data   0,   0
```

Pop Menus Source Code

```
GN 5 sys 700
OG 6 .opt oo
   10 ;**************************************
PJ 20 ;**      window and pop menu manager     **
AG 30 ;**           by darren james spruyt     **
GF 40 ;**                                      **
OE 50 ;**(c) 1985 by                           **
AF 60 ;**                  darren james spruyt **
   90 ;**************************************
AK 1000 ;define variables
DO 1010 *          =    $c000
II 1015 lpickarea  =    $02
IN 1020 xby4       =    $0256
BO 1030 yby4       =    $0255
CD 1040 avail      =    $0258
LN 1050 line       =    $0257
GD 2000 start      =    *
HH 2010            jmp  tsprite      ;pick areas on
FD 2020            jmp  anpickarea   ;add new pick area
EH 2025            jmp  popmenu      ;pop menu
DL 2030            jmp  dpickarea    ;delete pick area
NP 2040            jmp  pareasoff    ;pick areas off
JO 5000 f1         rts
JN 10000 popmenu   =    *            ;popmenu entry
NF 10002           ldy  #0
EH 10004           jsr  getval       ;get four
BH 10006           bmi  f1           ;parameters
KE 10008           lda  pickheight   ;for input
FL 10009           cmp  #3           ;and check
LJ 10010           bcc  f1           ;for minimum
```

```
GM 10011          lda  pickwidth  ;width and
DE 10012          cmp  #3         ;height
OB 10013          bcc  f1
NM 10014          sty  avail      ;set avail flg
MB 10016          jsr  getparam
HN 10018          stx  color      ;get color
FL 10020          jsr  $aefd      ;check comma
EH 10022          jsr  $ad9e      ;eval input
PB 10024          ldy  $65
OL 10026          lda  $64
OA 10028          jsr  $b6db      ;cln desc stk
NH 10030          ldy  #2
GA 10032          lda  ($64),y
BN 10034          sta  $48        ;get add hi
PO 10036          dey
MA 10038          lda  ($64),y
EO 10040          sta  $47        ;get add lo
LN 10050          lda  $47
NI 10052          bne  en0        ;dec address
EI 10054          dec  $48        ;by one
GP 10056 en0      dec  $47        ;
FF 10090          lda  #0
DE 10100          jsr  copy1      ;copy section
DC 10200          ldy  #0
BC 10210          jsr  makep      ;make pntr
AE 10220          ldx  pickheight
HC 10225          jsr  colorline
CO 10230          ldy  pickwidth
LL 10240          dey
GG 10250          lda  #$6e       ;"C= and S"
AJ 10260          sta  ($fb),y
JN 10270          dey
DC 10272          bmi  en1
AC 10280          lda  #$40       ;"Shift and *"
EM 10290 en2      sta  ($fb),y
HP 10300          dey
BE 10302          bmi  en1
PD 10310          bne  en2
DB 10320          lda  #$70       ;"C= and A"
GN 10330          sta  ($fb),y
OL 10340 en1      stx  temp
LG 10350 en1a     lda  #$28       ;
ML 10360          ldx  #$fb       ;
CN 10370          jsr  add        ;increase pntr
NL 10375          jsr  colorline
CJ 10380          ldx  temp       ;line cntr
NE 10390          dex
HJ 10400          beq  en3        ;exit if done
OK 10402          cpx  #1         ;
OL 10404          beq  en6
FP 10410          ldy  #0
IA 10420          lda  #$5d       ;"Shift and -"
KD 10430          sta  ($fb),y
MP 10440 en4      iny
AN 10450          cpy  pickwidth
DP 10460          beq  en5
DA 10470          lda  ($47),y    ;get char
ME 10475          jsr  corrascii
MG 10480          sta  ($fb),y    ;to screen
CD 10490          jmp  en4
FC 10500 en5      dey
CG 10510          lda  #$5d       ;"Shift and -"
PI 10515          sta  ($fb),y
PB 10520          stx  temp
OK 10530          lda  pickwidth
NK 10540          sec
BH 10550          sbc  #$02
HO 10560          ldx  #$47
KH 10570          jsr  add        ;inc pntr
HI 10580          jmp  en1a
PC 10640 en6      ldy  pickwidth  ;
FF 10650          dey
DB 10660          lda  #$7d       ;"C= and X"
KC 10670          sta  ($fb),y
AL 10680          lda  #$40       ;"Shift and *"
PO 10690 en8      dey
JO 10700          beq  en7
PN 10710          bmi  en3
MF 10720          sta  ($fb),y
OC 10730          jmp  en8
OP 10740 en7      lda  #$6d       ;"C= and Z"
KH 10750          sta  ($fb),y
OH 10760 ;
EF 10800 en3      ldx  #1         ;set to top
PH 10810 ep3      jsr  revline
IG 10900 ep2      stx  temp       ;save line
NF 10905          jsr  $ffe4      ;get char
EG 10907          ldx  temp
DA 10910          cmp  #"         ;up?
CK 10920          bne  ep1
HG 10930 ;up
ML 10940          cpx  #1         ;at topprint
AK 10950          beq  ep2        ;yes
FC 10960          jsr  revline    ;unrevrs line
BJ 10970          dex
BJ 10980          jmp  ep3        ;up 1
HH 11000 ep1      cmp  #"         ;down?
FA 11010          bne  ep4
IL 11012          txa
MH 11014          clc
JO 11016          adc  #2
BF 11020          cmp  pickheight ;at bottom
AP 11030          beq  ep2        ;yes
FH 11040          jsr  revline    ;unrevrs line
GA 11050          inx
PP 11060          jmp  ep3        ;increase line
FC 11100 ep4      cmp  #$0d       ;is a return
PD 11110          bne  ep2        ;nope
NC 11120          stx  $0257      ;set line num
KJ 11130          lda  #$80       ;copy back
MM 11140          jsr  copy1      ;data to sc
BC 11150          lda  #1         ;release pntr
OF 11160          sta  avail
HC 11170          rts             ;back to basic
BG 11499 ;
HJ 11500 colorline =   *
CO 11510          ldy  pickwidth
FI 11520          jsr  imagepntrs ;backup pntrs
FM 11530          dey
NN 11540          lda  color      ;set line
AA 11550 cl1      sta  ($fd),y    ;according
DO 11560          dey
JP 11570          bpl  cl1        ;finish
IC 11580          rts
DI 12000 copy1    sta  dir
DG 12005          lda  #0         ;set ($22) to
OK 12010          sta  $22
GO 12020          lda  #$b0       ;$b000
FM 12030          sta  $23
LE 12032          ldy  #0
KE 12034          jsr  makep      ;make address
HI 12040          lda  $01
ND 12050          and  #%11111110
LO 12060          sei             ;lock irqs
JN 12070          sta  $01        ;open the rom
ON 12080          jsr  imagepntrs
MK 12100 ;transfer from ($fb) to ($22)
CK 12110          ldx  pickheight
JM 12115 ep9      ldy  pickwidth
DB 12120          dey
AH 12122 epb      lda  dir
IK 12124          bpl  epa
DC 12126          lda  ($22),y    ;copy from memory
LN 12127          sta  ($fb),y    ;to screen
KC 12128          lda  ($24),y
FO 12129          sta  ($fd),y
KK 12130          jmp  ep8
LB 12132 epa      lda  ($fb),y    ;copy from screen
FP 12134          sta  ($22),y    ;to memory
OK 12136          lda  ($fd),y
CH 12138          sta  ($24),y
JK 12150 ep8      dey
EG 12160          bpl  epb        ;finish line
GE 12200 ;inc pntrs
JL 12210          stx  temp
EN 12220          lda  #$28       ;add $28 to $fb
PM 12230          ldx  #$fb
KD 12240          jsr  add
CP 12250          lda  #$28       ;add $28 to $fb
BH 12260          ldx  #$22
IF 12270          jsr  add
AN 12275          jsr  imagepntrs ;copy pntrs
BM 12280          ldx  temp
JL 12290          dex
IK 12300          bne  ep9        ;finish all lines
FJ 12310          lda  $01
FL 12320          ora  #%00000001
LI 12330          sta  $01        ;close roms
CM 12340          cli
JN 12400          rts             ;and finish up
PD 12900 imagepntrs =  *
CD 12905          lda  $22        ;backup ($22) to
ID 12910          sta  $24
MP 12915          lda  $23
HE 12920          ora  #$04
FD 12925          sta  $25        ;($24)
HN 12930          lda  $fb        ;backup ($fb) to
JK 12935          sta  $fd
NG 12940          lda  $fc
OH 12945          and  #$03
JJ 12950          ora  #$d8
HC 12955          sta  $fe        ;($fd)
MI 12960          rts
JH 13000 revline  =    *
IN 13020          ldy  #0         ;
ED 13030          txa             ;.x holds line
DJ 13032          sta  temp
GG 13040          clc
BK 13050          adc  pareay     ;add pick offset
KM 13055          tax
GA 13060          jsr  makep1     ;make pntr
HF 13070          ldy  pickwidth  ;get width of line
DN 13080          dey
FN 13082          dey
HG 13090 rvl1     lda  ($fb),y
HO 13100          eor  #$80       ;reverse char
KG 13110          sta  ($fb),y    ;back to sc
LP 13120          dey
HP 13130          bne  rvl1       ;finish line
FF 13135          ldx  temp       ;restore .x
AE 13140          rts
NA 14999 ;
FK 15000 dpickarea =   *
OF 15010          jsr  getparam   ;get pick are
MK 15020          cpx  #17
FH 15030          bcs  ep7        ;error so exit
FL 15050          lda  #0
HC 15060          sta  pareasopen,x ;delete with 0
EF 15070          rts             ;done
MP 15080 ep7      lda  #$ff       ;error return
ON 15090          rts
LO 15989 ;
EP 15999 pareasoff =   *          ;turn areas off
NB 16000          lda  $d015
IL 16010          and  #%01111111
MA 16020          sta  $d015      ;turn off sprite
HD 16030          sei
OE 16040          lda  #$ea
PN 16050          sta  $0315      ;reset irq
MO 16060          lda  #$31
KE 16070          sta  $0314      ;vector and
OF 16080          cli
LA 16090          rts             ;exit
NN 16999 ;
IK 19000 add      =    *          ;add routine
IL 19010          clc
PF 19020          adc  $00,x      ;add value in .a
IB 19030          sta  $00,x
DH 19040          bcc  add1       ;to indirect
BB 19050          inc  $01,x
MK 19060 add1     rts  ;at $00,x
DP 19069 ;
IE 19100 corrascii =   *          ;correct ascii
KB 19110          cmp  #$40
MB 19120          bcc  cr1        ;characters
JP 19130          sbc  #$40
BE 19140 cr1      cmp  #$80       ;before placing
EK 19150          bcc  cr2
FE 19160          sbc  #$40       ;on the screen
JB 19170 cr2      rts
FJ 19999 ;
DJ 20000 anpickarea =  *
JB 20010          ldy  #16
CG 20014 an0      lda  pareasopen,y
HD 20016          beq  an1
PO 20020          dey
FC 20022          bne  an0
BA 20040          lda  #$fe
BK 20060 ep6      rts
OL 20100 an1      =    *
OI 20110          jsr  getval
OK 20120          bmi  ep6
BP 20130          sta  pareasopen,y
IJ 20140          rts
HJ 20200 getval   jsr  getparam
KL 20210          cmp  #40
ED 20220          bcs  error
BG 20230          sta  pareax,y
MA 20240          jsr  getparam
NO 20250          cmp  #25
MF 20260          bcs  error
LI 20270          sta  pareay,y
ED 20280          jsr  getparam
DH 20285          beq  error
PG 20290          sta  pickwidth,y
KL 20292          clc
FF 20294          adc  pareax,y
```

```
AB 20296           cmp  #40
CI 20298           bcs  error
IE 20300           jsr  getparam
HI 20305           beq  error
BH 20307           sta  pickheight,y
MM 20310           clc
BI 20312           adc  pareay
NC 20314           cmp  #25
EJ 20316           bcs  error
NE 20320           lda  #1
AG 20340           rts
OF 20350 error     =    *
FE 20360           lda  #$ff
II 20380           rts
KP 29000 irqentry  =    *
CD 29100           lda  #>retcall   ;
CD 29110           pha
HD 29120           lda  #<retcall   ;set fake irq
GE 29130           pha
CJ 29140           php              ;call data
KF 29150           pha
EG 29160           pha
OG 29170           pha
HJ 29200           jmp  $ea31       ;do irq
LB 29500 retcall   =    *           ;back here
AF 29510           lda  avail       ;is ok
DC 29520           bne  rr1         ;yes
PG 29530 ex1       jmp  $febc       ;finish irq
LG 29600 rr1       lda  $9d         ;in basic
NJ 29610           bmi  ex1         ;nope
EH 29612           lda  $cc         ;cursor on
BG 29614           beq  ex1         ;yes - exit
IM 29620           ldy  $c6
DO 29630           lda  $0276,y     ;get last chr
FB 29640           ldx  #3
PM 29650 af0       cmp  tablea,x    ;check against
AN 29660           beq  af1         ;table
NJ 29670           dex
BP 29680           bpl  af0
MO 29690           bmi  af2
GM 29700 af1       dec  $c6         ;delete from
BC 29710 af2       lda  $cb         ;buffer + get
MJ 29720           cmp  #$07
PB 29730           beq  af3
MJ 29740           cmp  #$02
AE 29750           bne  ex1
CL 29760 af3       and  #$01
JE 29763           ldy  $028d
JL 29765           beq  cup
NK 29770           cpy  #3
FH 29780           bcs  ex1
HC 29790           ora  #$02
AA 29800           jmp  cup
NJ 30100 cup       cmp  #03         ;up
CI 30110           bne  cdown
FE 30130           ldy  ypos
NA 30140           beq  end         ;at top - ex
MH 30145           dey
DO 30150           sty  ypos        ;decrease
LA 30160           jmp  end
KK 30200 cdown     cmp  #1          ;down
KJ 30210           bne  cleft
JK 30230           ldy  ypos
KD 30240           cpy  #99
KA 30250           bcs  end         ;at bottom -ex
EB 30260           iny
IH 30270           sty  ypos        ;increase
DI 30280           jmp  end
BM 30300 cleft     cmp  #02         ;left
GL 30310           bne  cright
MM 30320 ;left
MA 30330           ldy  xpos
JF 30340           beq  end         ;at left - ex
EE 30345           dey
KK 30350           sty  xpos        ;decrease
DN 30360           jmp  end
IO 30400 cright    cmp  #00         ;right
MM 30410           bne  end
HD 30420 ;right
AH 30430           ldy  xpos
HA 30440           cpy  #159        ;at right-ex
LA 30450           bcs  end
MN 30460           iny
PD 30470           sty  xpos        ;increase
KO 30500 end       =    *
GM 30510           lda  $d010
GG 30520           and  #%01111111
BL 30530           sta  $d010       ;zero high bit
OH 30540           lda  xpos
BA 30550           asl
GG 30560           bcc  ck6         ;x *2
LG 30565           jsr  setbit      ;set if ness.
NE 30570 ck6       clc
BP 30580           adc  #24
FE 30590           bcc  ck7         ;add offset
OI 30600           jsr  setbit      ;set if ness.
NF 30610 ck7       sta  $d00e       ;set lo byte
PM 30620           lda  ypos        ;get ypos
BF 30630           asl
HC 30640           adc  #50
HA 30650           sta  $d00f       ;set it
KE 31000           lda  xpos
LP 31010           lsr
GB 31020           lsr              ;/ xpos by 4
KL 31030           sta  xby4        ;to yield char
LG 31040           lda  ypos        ;positions
DC 31050           lsr
BE 31060           lsr              ;/ ypos by 4
MG 31070           sta  yby4        ;as above
JF 31102           ldy  #16         ;
PL 31104 ck4       lda  pareasopen,y
KI 31105           beq  ck3
FK 31110           lda  xby4
CP 31120           cmp  pareax,y
BI 31130           bcc  ck3         ;to the left
LN 31150           sbc  pareax,y
MO 31160           cmp  pickwidth,y
OB 31170           bcs  ck3         ;to the right
MO 31180           lda  yby4
KD 31190           cmp  pareay,y
DL 31200           bcc  ck3         ;above area
JB 31210           sbc  pareay,y
JB 31220           cmp  pickheight,y
HD 31230           bcs  ck3         ;below bottom
KJ 31240           cpy  lpickarea   ;was last
AI 31250           beq  ck1         ;yes - no prob
DK 31260           sty  temp1       ;save new on
LL 31270           ldy  lpickarea   ;reverse last
BA 31280           jsr  revarea     ;pick area
BN 31290           ldy  temp1       ;get new area
GA 31300           sty  lpickarea   ;store cur pic
MO 31310           jsr  revarea     ;reverse area
KJ 31320 ck1       jmp  $febc       ;exit irq
PI 31350 ck3       dey              ;do all
EI 31370           bne  ck4         ;open picks
OK 31380           ldy  lpickarea   ;if none revrs
PK 31390           jsr  revarea     ;last area
DJ 31400           lda  #0
CP 31410           sta  lpickarea   ;set to 0
IN 31420           jmp  $febc       ;done irq
FH 31999 ;
OD 32000 revarea   =    *           ;
JL 32002           tya
OA 32004           beq  dn1
DF 32020           jsr  makep       ;make pntr
EL 32120           ldx  pickheight,y
BH 32130           lda  pickwidth,y
HD 32140           sta  temp
JP 32145           dec  temp
KP 32150 rv0       ldy  temp
ME 32160 rv1       lda  ($fb),y     ;get char
JJ 32170           eor  #$80        ;reverse
IO 32180           sta  ($fb),y     ;back to sc
JH 32190           dey
CJ 32200           bpl  rv1         ;finish line
AL 32210           lda  $fb
CF 32220           clc
BC 32230           adc  #$28
MA 32240           sta  $fb
HO 32250           bcc  rv2
LO 32260           inc  $fc         ;increase pntr
CF 32270 rv2       dex
GL 32280           bne  rv0         ;finish lines
LE 32290 dn1       rts
HP 38000 makep     =    *           ;make pointer
FP 38010           ldx  pareay,y
CG 38020 makep1    lda  $d9,x       ;at ($fb),
LH 38030           and  #$03
LM 38040           ora  $0288       ;to point
BM 38050           sta  $fc
HB 38060           lda  $ecf0,x     ;to screen line
MC 38070           clc
LD 38080           adc  pareax,y    ;according to
GO 38090           sta  $fb
MA 38100           bcc  pn1         ;pick area in
PN 38110           inc  $fc
FP 38120 pn1       rts              ;.y
OK 39000 setbit    =    *           ;set msb of
JN 39005           pha
GF 39010           lda  $d010       ;sprite pos
CA 39020           ora  #%10000000
ME 39030           sta  $d010
DA 39035           pla
MG 39040           rts
GI 40000 tsprite   =    *           ;copy sprite
KD 40010           ldy  #63
OO 40020 ts1       lda  spritedata,y
IA 40030           sta  $03c0,y     ;to low memory
DC 40040           dey
DK 40050           bpl  ts1
LH 41010           sei              ;lock irq's
FG 41020           lda  #>irqentry
CA 41030           sta  $0315
NH 41040           lda  #<irqentry
LL 41050           sta  $0314       ;set vector
CP 41060           cli
PO 41100           lda  #128
NL 41110           sta  53248+21    ;turn on
MC 41120           lda  #15
IO 41130           sta  $07ff       ;set pic loc
OP 41135           sta  avail       ;set avail flg
PJ 41140           lda  #0
FK 41150           sta  pareasopen  ;zero open
FP 41152           sta  lpickarea   ;set last pick
CD 41154           sta  xpos
HA 41156           sta  ypos        ;start pos
DL 41160           lda  #0
OA 41170           ldy  #16         ;clear flags
EH 41180 ts2       sta  pareasopen,y
BK 41190           dey
GN 41200           bpl  ts2         ;for pick areas
FI 41210           lda  #24
CG 41220           sta  $d00e       ;set start
DJ 41230           lda  #50
AA 41240           sta  $d00f
LG 41410           rts              ;sprite pos
FH 42000 getparam  =    *
FC 42010           sty  temp
FL 42020           jsr  $aefd       ;check comma
GI 42030           jsr  $b79e       ;0-255 parameter
FA 42040           ldy  temp
AB 42050           txa              ;return in .a
ID 42060           rts
LI 50000 spritedata =   *
KB 50010 .byte%11111110,%00000000,%00000000
KB 50020 .byte%11100000,%00000000,%00000000
HC 50030 .byte%11110000,%00000000,%00000000
DD 50040 .byte%11011000,%00000000,%00000000
LD 50050 .byte%11001100,%00000000,%00000000
DE 50060 .byte%11000110,%00000000,%00000000
KE 50070 .byte%00000011,%00000000,%00000000
CF 50080 .byte%00000001,%00000000,%00000000
AH 50100 .byte 0,0,0
KH 50110 .byte 0,0,0
EI 50120 .byte 0,0,0
OI 50130 .byte 0,0,0
IJ 50140 .byte 0,0,0
CK 50150 .byte 0,0,0
MK 50160 .byte 0,0,0
GL 50170 .byte 0,0,0
AM 50180 .byte 0,0,0
KM 50190 .byte 0,0,0
EN 50200 .byte 0,0,0
ON 50210 .byte 0,0,0
IO 50220 .byte 0,0,0
CB 50500 tablea    .byte $11,$1d,$91,$9d
MO 60000 ;internal variables
DM 60010 ypos      *=*+1
FN 60030 xpos      *=*+1
DI 60050 pareasopen =    *
AN 60060 .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
OJ 60080 pareax    =    *
OO 60090 .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
FL 60100 pareay    =    *
CA 60110 .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
NO 60120 pickwidth =    *
GB 60130 .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
II 60140 pickheight =   *
KC 60150 .byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
NE 60200 temp      *=*+1
EN 60220 dir       *=*+1
MA 60240 temp1     *=*+1
BN 60500 color     *=*+1
```

Dvorak Keyboard For The Commodore 64

Donald P. Maple
Calgary, Alberta

. . .Christopher Latham Sholes came up with the QWERTY layout which places the most-used letters as far apart as possible. . .

The following program will redefine the Commodore 64 keyboard to the Dvorak layout. This layout facilitates more efficient typing and reduces fatigue.

Contrary to popular belief, the keyboard layout that the majority of us face when using our computers is neither the first, nor only keyboard layout available. As we all know, this keyboard layout, ingeniously nicknamed "QWERTY", is an atavism that computer users have inherited from the typewriter.

The invention of both the typewriter and QWERTY keyboard can be credited to a gentleman by the name of Christopher Latham Sholes. Often times his name will appear in reference to the legendary QWERTY keyboard under the pseudonym of the "Sholes" keyboard. This invention came about in the 1860's, with a patent being granted for both the typewriter and keyboard layout together. In truth, prior to Sholes' typewriter there were about 50 other typewriter patents in existence. Curiously, though, the current Sholes layout is actually an *improved* version of his early attempts. Namely, the original layouts proved to be *too good* in that they enabled the operators to achieve considerable typing speed. Contrary to what one would expect, this was an unwelcomed feature due to the inferior mechanics of the typewriter. The advantages of high speed typing were quickly wasted due to the constant jamming of the keys. To "correct" this shortcoming, Sholes came up with the current QWERTY layout which places the most-used letters as far apart as possible on the keyboard, thus limiting even the most proficient typist.

At the beginning of this century, however, several research projects were undertaken to speed up both learning to type and the typing speed itself. This is where August Dvorak (pronounced Dvorzhak) came in. He invented the Dvorak Simplified Keyboard (see Fig 1) which was designed to increase typing speed and decrease fatigue by altering the arrangement of the keys. This new arrangement was based on word sampling and observance of often used words as well as the most common three letter combinations (see Fig 2 for the Top Ten). The purpose of the new layout was to minimize the movements of the users fingers. This, again, prompted numerous studies in comparing the efficiency of Dvorak against QWERTY; the results showed an improvement between 20% to 50%. Current belief is that the actual speed increase is in the 20% range but the elimination of fatigue results in a 50% long term improvement.

In spite of these impressive statistics, the Dvorak layout has been very slow in coming into general acceptance. It is, however, most efficient with the English language and the increased comfort and speed more than make up for having to relearn typing all over again. So, while waiting for voice recognition systems to eliminate the use of keyboards altogether, here goes. . .

Ladies and Gentlemen, Start Your Keyboards!

To get some hands on experience, first type in the accompanying Basic loader program named "DVORAK.LDR". Make sure the program is saved before running it. The Basic loader calculates a checksum to insure that all data is correct. If the checksum proves inaccurate, the message "DATA ERROR" will show up to indicate the obvious. If, on the other hand, all goes well, you will find that your keyboard has been redefined along the lines of the DVORAK standard.

There are some minor variations to the regular DVORAK layout due to the uniqueness of the Commodore 64. These are as follows:

Shifted 6 will produce a '£' sign instead of the standard DVORAK cent sign, which is absent with the Commodore 64.

Shifted '–' will produce the '–' instead of underdash which, again, is absent with the Commodore 64.

Note also that the graphics characters associated with the keys that have been moved, have also moved. This applies only to the the graphic characters obtained while the SHIFT key is

down. The ones obtained by pressing the Commodore key have not moved. This is because only two of the four keyboard tables have been redefined. The reason for moving only the unshifted and shifted tables is because these two tables contain the alphanumeric characters which is where the DVORAK keyboard layout is at its strongest. In other words, this layout is perceived as most useful in text based applications which use graphics characters very sparingly if at all.

Finally, there has been one minor modification to the KERNAL so that pressing STOP/RESTORE will not revert back to QWERTY layout. This is so that if the program you happened to be running locks–up, STOP/RESTORE will come to the rescue while maintaining the Dvorak keyboard.

How It's Done

Before we get into the ins and outs, a few words about how the values for individual keys are arrived at in the Commodore 64. If you count all the keys on your keyboard, including the function and all other special keys, the total number is 65. One key, the SHIFT LOCK, is actually the same as the left SHIFT key since it mechanically holds this key down. So the actual number of keys the system can "see" is 64. These keys, which in effect are just simple switches, are arranged into an 8 × 8 matrix. (see Fig 3) This matrix is connected to two registers of the CIA#1 chip. One register, $DC00 (56320), connects to the keyboard column, while the other, $DC01 (56321), connects to the keyboard row. Without going into too much detail, these two registers eventually yield the key number. This number is used as an index to obtain the actual key value from the key table. This value is what we are all familiar with when we sample the keyboard using the GET statement in BASIC.

There are, as a matter of fact, four different key tables. The table which is accessed is determined by the operating system based on whether the SHIFT, Commodore or CTRL keys are pressed in addition to "regular" keys. The four tables are at the following addresses:

1. $EB81–$EBC1 (60289–60354) no special keys pressed
2. $EBC2–$EC02 (60355–60419) SHIFT pressed
3. $EC03–$EC43 (60420–60484) Commodore key pressed
4. $EC78–$ECB8 (60536–60600) CTRL key pressed

Each table contains 65 entries, but this is only to insure that if the search is unsuccessful, the 65th value will be returned. This value is $FF (255) and you will find that it is also used in place of any invalid or unused key combination.

The "DVORAK.LDR", as mentioned earlier, pokes a short machine language routine in the cassette buffer. This routine basically does two things. First, the complete contents of the operating system ROM at locations $E000 – $FFFF (57344 – 65535) are copied into the RAM below. BASIC, which is not modified, is also copied. This is due to the fact that it is impossible to switch the Kernal ROM out without losing BASIC ROM as well. This is accomplished by logically ANDing location 1 with #$FD (253). Secondly, once the ROMs are copied, the keyboard layout tables are reloaded. Only the first two tables are modified because that is where the alphanumeric characters reside.

This program can also be used to experiment with different keyboard layouts. The data lines containing the key values correspond to the keyboard matrix in Fig 3. Simply locate the key to be modified and place the key code in the corresponding place in the DATA statements.

Let us, for example, redefine the F1 key to perform a 'clear screen'. A quick check in the keyboard matrix (Fig 3) reveals that the F1 key is located in row 1, column 5. The corresponding location in the DATA statements, line 270, entry 5, currently contains value of $85 (133). This is indeed the code for F1 as a quick check in the Programmer's Reference Guide will verify. If you do not have the above guide, try this short program:

```
10 get a$: if a$ = " " then 10
20 print asc(a$)
```

It will wait for a key to be pressed and then show the ASCII value of the key in decimal. So, all that is now left to do is to get the value for a 'clear screen', which happens to be $93 (147), and place this value as the fifth entry on line 270. Note that this will upset the checksum which accounts for the "DATA ERROR" message when the program is run. Since this was intentional, ignore it and enable the layout by typing:

```
SYS 820
```

Now press F1 and sure enough, the screen has been cleared!

Or how about changing the character colour by using one of the F–keys? Try putting a $1C (28) as the sixth entry on line 270. This will promptly make F3 change the character colour to red!

Conclusion

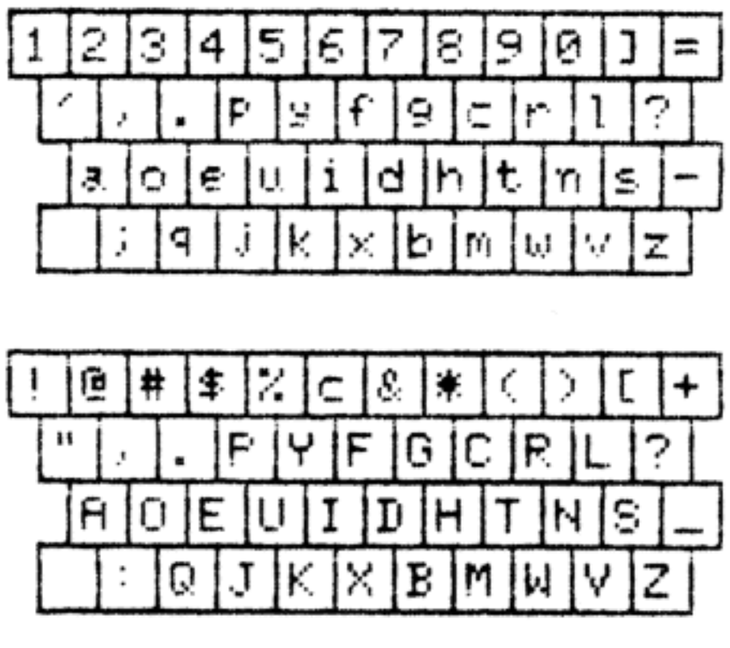
This article and the accompanying program have introduced some keyboard concepts as applied to Commodore 64. If you have any questions or suggestions, you can contact me either through this magazine or directly at the address listed below.

Donald P. Maple
P.O.Box 23, Station M
Calgary, Alberta
Canada T2P 2G9

DVORAK Basic Loader

```
GI  10 rem save "0:dvorak.ldr",8
FK  100 for i=820 to 1017: read x$
AM  110 h=asc(left$(x$,1))-48: if h>9 then h=h-7
AP  120 l=asc(right$(x$,1))-48: if l>9 then l=l-7
EK  130 x=h*16+l: poke i,x: ch=ch+x: next
GJ  140 if ch<>25301 then print "** data
        error **": end
OI  150 sys820: print "** dvorak keyboard
        enabled **": end
EB  160 :
MP  170 data a0, 00, 84, fb,  84, fd,  a9, e0
PF  180 data 85, fc,  a9, a0, 85, fe,  b1, fb
ME  190 data 91, fb,  b1, fd,  91, fd,  88, d0
CN  200 data f5,  e6, fe,  e6, fc,  d0, ef,  a5
BH  210 data 01, 29, fd,  85, 01, a9, 77, 85
EG  220 data fb,  a9, 03, 85, fc,  a9, 80, 85
ML  230 data fd,  a9, eb, 85, fe,  a0, 81, b1
CJ  240 data fb,  91, fd,  88, d0, f9,  a9, e5
IB  250 data 8d, d6, fd,  60
PF  260 rem *** unshifted keys ***
FL  270 data 14, 0d, 1d, 88, 85, 86, 87, 11
GO  280 data 33, 2c, 41, 34, 3b, 4f,  2e, 01
MJ  290 data 35, 50, 45, 36, 4a, 55, 59, 51
KK  300 data 37, 46, 49, 38, 58, 44, 47, 4b
PL  310 data 39, 43, 48, 30, 4d, 54, 52, 42
AF  320 data 5d, 4c, 4e, 3d, 56, 53, 3f,  57
FE  330 data 5c, 2a, 2d, 13, 01, 3d, 5e, 5a
AI  340 data 31, 5f,  04, 32, 20, 02, 27, 03
OE  350 data ff
NG  360 rem *** shifted keys ***
LL  370 data 94, 8d, 9d, 8c, 89, 8a, 8b, 91
MG  380 data 23, 2c, c1, 24, 3a, cf,  2e, 01
FN  390 data 25, d0, c5, 5c, ca, d5, d9, d1
PM  400 data 26, c6, c9, 2a, d8, c4, c7, cb
EO  410 data 28, c3, c8, 29, cd, d4, d2, c2
KF  420 data 5b, cc, ce, 2b, d6, d3, 3f,  d7
PP  430 data a9, c0, 2d, 93, 01, 3d, de, da
GC  440 data 21, 5f,  04, 40, a0, 02, 22, 83
CL  450 data ff
```

DVORAK LAYOUT



LOWER CASE

UPPER CASE

Dvorak keyboard/MaPle/FIGURE 1

WORD OCCURRENCE TABLE

PLACE	WORD	3LETTR
1	THE	THE
2	OF	AND
3	AND	ING
4	TO	IUN
5	A	ENT
6	IN	TIO
7	THAT	EOP
8	IS	HER
9	WAS	TER
10	HE	ATI

IVORAK KEYBOARD/MAPLE/FIGURE 2

KEYBOARD DECODE TABLE

	1	2	3	4	5	6	7	8
1	DEL	RETRN	CR.RT	F7	F1	F3	F5	CR.DN
2	3	W	A	4	Z	S	E	L.SHF
3	5	R	D	6	C	F	T	X
4	7	Y	G	8	B	H	U	V
5	9	I	J	0	M	K	O	N
6	+	P	L	-	.	:	@	,
7	£	*	;	HOME	R.SHF	=	↑	/
8	1	←	CTRL	2	SPACE	C=	Q	STOP

DVORAK KEYBOARD/MAPLE/FIGURE 3

Screenboard For The Commodore 64

David Tomblin
Parksville,
British Columbia

Probably the most rewarding endeavour a computer hobbyist can pursue is using the computer to help other people. The following program is just such an endeavour.

In Transactor Volume 5 Issue 4 there was an article called "Helping The Handicapped". If you missed it, the article concerned an attachment for a wheelchair consisting of pushbuttons on a board that could be used as a joystick simulator. I read the article while going through my back issues of the "T" and was inspired to write the program you see here.

"Screenboard" is what I call this program and the name describes it well. Because of certain physical handicaps, the keyboard of a computer is just another obstacle for some people. What Screenboard does is make keyboard operation as easy as using a joystick.

When executed, the program will display keyboard characters in the top portion of the screen. A cursor is moved around inside the keyboard 'window' by moving a joystick plugged into port 2. A key is selected by placing the cursor over the desired character or abbreviation (eg. CUP for cursor–up) and pressing the fire button.

On a more technical side, the program is executed through the IRQ vector so it may not be compatible with all software. It also works through the 'test stop key' vector at $0328. It may be disabled by hitting the restore key.

The PAL source code is included for anyone who wishes to see how it works or modify it. The basic loader, for those who don't is rather lengthy to type in but probably worth the effort (thank goodness for Transactor disks).

I hope this program will open up the fascinating world of computers to many handicapped people. I also hope other computer users will use their skills to help people less fortunate than themselves. Thanks to Phillip J. Honsinger for the inspirational article.

Screenboard BASIC Loader Program

```
ND | 10 rem* data loader for " screenboard " *
LI | 20 cs = 0
LF | 30 for i = 49152 to 49903:read a:poke i,a
DH | 40 cs = cs + a:next i
GK | 50 :
CD | 60 if cs<>79077 then print " !data error! " : end
DD | 70 sys 49152
AF | 80 end
IN | 100 :
DH | 1000 data  76,  18, 192,  14,   0,   0,   0,   0
AM | 1010 data   0,   0,   0,   0,   0,   0,   0,   0
GC | 1020 data   0,   0, 120, 169,  89, 141,  20,   3
EP | 1030 data 169, 192, 141,  21,   3, 173, 143,   2
KE | 1040 data 141,  14, 192, 173, 144,   2, 141,  15
AG | 1050 data 192, 173,  40,   3, 141,  16, 192, 173
BD | 1060 data  41,   3, 141,  17, 192, 169,  74, 141
DK | 1070 data  40,   3, 169, 192, 141,  41,   3,  88
KE | 1080 data  96, 173,  12, 192, 141, 141,   2, 108
GG | 1090 data  14, 192, 173,  13, 192, 240,   7, 133
MD | 1100 data 145, 169,   0, 141,  13, 192, 108,  16
OA | 1110 data 192,  32, 118, 193,  56,  32, 240, 255
EI | 1120 data 224,  6, 176,   6, 162,   6,  24,  32
NB | 1130 data 240, 255,  32, 113, 192,  76,  49, 234
IB | 1140 data  96, 173,   0, 220,  41,  31,  73,  31
PK | 1150 data 133, 253, 208,   8, 169,   0, 141,   6
KG | 1160 data 192,  76, 233, 192, 173,   6, 192, 208
MD | 1170 data  96, 169,   1, 141,   6, 192, 169,  16
LK | 1180 data  36, 253, 208,  86, 165, 253,  41,   1
MF | 1190 data 240,   2, 162, 216, 165, 253,  41,   2
JK | 1200 data 240,   2, 162,  40, 165, 253,  41,   4
FC | 1210 data 240,  14, 173,   4, 192, 201, 120, 144
JB | 1220 data   5, 162, 252,  76, 184, 192, 162, 254
HA | 1230 data 165, 253,  41,   8, 240,  14, 173,   4
LM | 1240 data 192, 201, 120, 144,   5, 162,   4,  76
HP | 1250 data 204, 192, 162,   2, 134, 139, 173,   4
HB | 1260 data 192,  24, 101, 139,  16,   4, 201, 200
OP | 1270 data 176,  15, 141,   4, 192, 141,   5, 192
HL | 1280 data 201, 120, 144,   5,  41, 252, 141,   5
PK | 1290 data 192,  96, 173,   4, 192, 201, 120, 176
JD | 1300 data   4,  74,  76, 253, 192,  56, 233, 120
IE | 1310 data  74,  74,  24, 105,  60, 170, 189,  26
PI | 1320 data 194, 201, 249, 176,  35, 224,  37, 176
JP | 1330 data  12, 189, 203, 194, 133, 203,  32,  65
NI | 1340 data 192,  32, 101, 193,  96, 164, 198, 204
```

```
FH 1350 data 137,   2, 176,   8, 189,  26, 194, 153
NE 1360 data 119,   2, 230, 198,  32, 101, 193,  96
DM 1370 data 201, 249, 208,   6, 169, 127, 141,  13
BA 1380 data 192,  96,  56, 233, 250,  74, 170, 169
CJ 1390 data   1, 105,   0, 133, 255, 189,   7, 192
NP 1400 data  69, 255,  37, 255, 157,   7, 192, 169
CF 1410 data   0, 141,  12, 192, 174,   7, 192, 240
MK 1420 data   2,   9,   1, 174,   8, 192, 240,   2
BN 1430 data   9,   4, 174,   9, 192, 240,   2,   9
PL 1440 data   2, 141,  12, 192,  96, 162,   2, 189
ID 1450 data   7, 192,  41,   2, 157,   7, 192, 202
CA 1460 data  16, 245,  32,  71, 193,  96, 173, 136
NL 1470 data   2, 133, 252, 169,   0, 133, 251, 141
AN 1480 data  10, 192, 141,  11, 192, 162,   0, 160
HO 1490 data   0, 169,  32, 145, 251, 189,  26, 194
EP 1500 data  41,  63, 204,   5, 192, 208,   2,   9
IP 1510 data 128, 200, 145, 251, 173,   3, 192, 153
AB 1520 data   0, 216, 200, 232, 224,  60, 144, 225
NP 1530 data 162,   0, 189, 106, 194, 201, 255, 208
PP 1540 data   3,  32, 239, 193,  41,  63, 133, 254
BF 1550 data 173,   5, 192, 201, 120, 144,  15, 152
FN 1560 data  24, 237,   5, 192, 201,   3, 176,   6
KE 1570 data 165, 254,   9, 128, 133, 254, 165, 254
ND 1580 data 145, 251, 173,   3, 192, 153,   0, 216
JJ 1590 data 200, 232, 224,  80,  48, 204, 162,  40
OP 1600 data 169,  64, 145, 251, 173,   3, 192, 153
KH 1610 data   0, 216, 200, 202, 208, 242,  96, 138
AG 1620 data  72, 174,  10, 192, 189,   7, 192,  10
EL 1630 data  10,  24, 109,  11, 192, 170, 189, 186
LL 1640 data 194, 133, 254, 238,  11, 192, 173,  11
MF 1650 data 192, 201,   3, 144,   8, 238,  10, 192
JL 1660 data 169,   0, 141,  11, 192, 104, 170, 165
FI 1670 data 254,  96,  65,  66,  67,  68,  69,  70
KA 1680 data  71,  72,  73,  74,  75,  76,  77,  78
CB 1690 data  79,  80,  81,  82,  83,  32,  84,  85
ND 1700 data  86,  87,  88,  89,  90,  48,  49,  50
JA 1710 data  51,  52,  53,  54,  55,  56,  57,  91
MP 1720 data  93,  32,  33,  34,  35,  36,  37,  38
KB 1730 data  39,  40,  41,  44,  46,  63,  58,  59
LG 1740 data  60,  61,  62,  64,  94,  32, 145,  17
CA 1750 data 157,  29, 148,  20,  19, 147,  43,  45
LC 1760 data  13, 249, 250, 251, 252, 253, 254, 255
BE 1770 data  42,  47,  32,  67,  85,  80,  32,  67
BH 1780 data  68,  78,  32,  67,  76,  70,  32,  67
DH 1790 data  82,  84,  32,  73,  78,  83,  32,  68
DK 1800 data  69,  76,  32,  72,  79,  77,  32,  67
IE 1810 data  76,  82,  32,  32,  43,  32,  32,  45
IH 1820 data  32,  32,  32,  82,  84,  78,  32,  83
NN 1830 data  84,  80,  32,  83,  72,  70,  58, 255
CI 1840 data 255, 255,  32,  67,  84,  76,  58, 255
GJ 1850 data 255, 255,  32,  67,  79,  77,  58, 255
NH 1860 data 255, 255,  32,  32,  42,  32,  32,  47
MJ 1870 data  32,  32,  79,  70,  70,  32,  79,  78
AG 1880 data  32,  32,  76,  79,  75,  32,   0,  40
FI 1890 data  80, 120, 160,  10,  28,  20,  18,  14
LJ 1900 data  21,  26,  29,  33,  34,  37,  42,  36
JL 1910 data  39,  38,  41,  62,  17,  13,  60,  22
PG 1920 data  30,  31,   9,  23,  25,  12,  35,  56
KK 1930 data  59,   8,  11,  16,  19,  24,  27,  32
```

Screenboard PAL Source Code

```
FD  100 sys700
DG  110 ;     "screenboard"
NF  120 ; joystick-controlled on-screen
FJ  130 ; keyboard
CA  140 ;
GD  150 ;original program by david tomblin
BK  160 ;this version jan86 -cz
AC  170 ;
NC  180 ;             from
MJ  190 ;     "The Transactor"
OD  200 ;
IE  210 ;
EE  220 .opt oo
HN  230 * = $c000
GG  240 ;
BC  250 chrout    =     $ffd2
CG  260 plot      =     $fff0
ML  270 screen    =     $fb
NM  280 joymask   =     $fd
KE  290 temp      =     $fe
DA  300 temp2     =     $ff
PO  310 xadd      =     $8b
BE  320 joy       =     56320
AM  330 ;
JG  340           jmp   start
EN  350 ;
JO  360 colour    .byte 14          ;screenboard colour
FL  370 sindex    .byte 0           ;screen index
OP  380 rvschar   .byte 0
EA  390 movflg    .byte 0
BN  400 keyflags  =     *
DJ  410           .byte 0           ;shift off/on/lock
KN  420           .byte 0           ;ctrl off/on/lock
NF  430           .byte 0           ;commdr off/on/lock
JP  440 togcnt    .byte 0
JF  450 tog2      .byte 0
MO  460 flagim    .byte 0
DD  470 stopflg   .byte 0
CD  480 olkvec    .word 0
IN  490 olstop    .word 0
KG  500 ;
EG  510 start     =     *
BK  520           sei
BO  530           lda   #<irqrtn    ;irq vector
HN  540           sta   $0314
PF  550           lda   #>irqrtn
MO  560           sta   $0315
JJ  570 ;save keyboard vector
MO  580           lda   $028f
PN  590           sta   olkvec
OO  600           lda   $0290
FL  610           sta   olkvec+1
PA  620 ;change the stop vector
LP  630           lda   $0328
NL  640           sta   olstop
AB  650           lda   $0329
KK  660           sta   olstop+1
FL  670           lda   #<newstop
LG  680           sta   $0328
FM  690           lda   #>newstop
AI  700           sta   $0329
EF  710           cli
ML  720           rts
AF  730 ;
KF  740 ;
LJ  750 newkbd    =     *           ;new keyboard setup rtn
BK  760           lda   flagim      ;shift/c=/ctrl
BO  770           sta   653
KO  780           jmp   (olkvec)
MI  790 ;
GJ  800 ;
CJ  810 newstop   =     *
IL  820           lda   stopflg
BH  830           beq   nostop
HB  840           sta   $91
ND  850           lda   #0
```

```
OB  860              sta   stopflg
JD  870 nostop       =     *
OD  880              jmp   (olstop)
AP  890 ;
KP  900 ;
CL  910 irqrtn       =     *
ON  920              jsr   drawscrn     ;draw screenboard
DC  930              sec
FD  940              jsr   plot         ;check cursor pos
IE  950              cpx   #6           ;see if it's in scrnbrd
MA  960              bcs   crsok        ;no it isn't
OC  970 ;yes it is, move cursor down
HC  980              ldx   #6
EF  990              clc
CO  1000             jsr   plot
FG  1010 crsok       =     *
KE  1020             jsr   scmove       ;scrnbd cursor move
PC  1030             jmp   $ea31
MP  1040             rts
AJ  1050 ;
KJ  1060 ;
IC  1070 scmove      =     *            ;move sc cursor row, col
DJ  1080             lda   joy
GA  1090             and   #31
HA  1100             eor   #31
LO  1110             sta   joymask
LA  1120             bne   scm0         ;stick moved
FF  1130             lda   #0
JP  1140             sta   movflg
AA  1150             jmp   nostor
DM  1160 scm0        =     *
AJ  1170             lda   movflg       ;moved last time
FO  1180             bne   nostor
DJ  1190             lda   #1
FD  1200             sta   movflg
MI  1210             lda   #16          ;fire button bit
CG  1220             bit   joymask
LD  1230             bne   keypush      ;enter key
PC  1240             lda   joymask
CD  1250             and   #1           ;check up
DE  1260             beq   scm1         ;not up, check down
IK  1270             ldx   #-40         ;up, subtract 40
MD  1280 scm1        =     *
BG  1290             lda   joymask
EC  1300             and   #2           ;check down
CB  1310             beq   scm2         ;not down, check left
AD  1320             ldx   #40          ;down, add 40
PG  1330 scm2        =     *
DJ  1340             lda   joymask
JE  1350             and   #4           ;check left
IG  1360             beq   scm3         ;not left, check right
ID  1370             lda   sindex
MO  1380             cmp   #120
MF  1390             bcc   sub2         ;subtract 2 for top rows
AM  1400             ldx   #-4          ;subtract 4 for bottom 2
JM  1410             jmp   scm3
DN  1420 sub2        =     *
FK  1430             ldx   #-2          ;left, subtract 2
ON  1440 scm3        =     *
BA  1450             lda   joymask
OM  1460             and   #8           ;check right
NF  1470             beq   scm4         ;not right
GK  1480             lda   sindex
KF  1490             cmp   #120
CJ  1500             bcc   add2
FD  1510             ldx   #4
LD  1520             jmp   scm4
CP  1530 add2        =     *
PE  1540             ldx   #2
EI  1550 ;
HF  1560 scm4        =     *
KN  1570             stx   xadd
KA  1580             lda   sindex
MK  1590             clc
AF  1600             adc   xadd
FD  1610             bpl   stornew      ;keep sb cursor
IN  1620             cmp   #200
IE  1630             bcs   nostor       ;in bounds
ME  1640 stornew     =     *
OI  1650             sta   sindex
FO  1660             sta   rvschar
OA  1670             cmp   #120
AL  1680             bcc   nostor
CM  1690             and   #%11111100
BC  1700             sta   rvschar      ;last 2 rows in 4s
BJ  1710 nostor      =     *
EK  1720             rts
ID  1730 ;
CE  1740 ;
OG  1750 keypush     =     *            ;enter key
OL  1760             lda   sindex
LM  1770             cmp   #120         ; " top 3 rows?
JP  1780             bcs   kps1         ;no
HM  1790             lsr   a            ;yes, just divide by 2
PG  1800             jmp   kps2
NH  1810 kps1        =     *
DP  1820 ;in bottom 3 rows
HK  1830             sec
DI  1840             sbc   #120
EF  1850             lsr   a
OF  1860             lsr   a
EM  1870             clc
EL  1880             adc   #klen
OM  1890 kps2        =     *
CO  1900 ;
BE  1910             tax
BA  1920             lda   keys,x       ;get key from table
OD  1930             cmp   #249
FO  1940             bcs   special      ;special character
CD  1950 normc       =     *            ;normal character
FI  1960             cpx   #37          ;check, alphanumerics
IG  1970             bcs   kbuf         ;no, use kbd buffer
NF  1980 ;yes, use custom keyboard trap
PK  1990             lda   keycodes,x   ;get key code
EI  2000             sta   203          ;last key
DM  2010             jsr   newkbd       ;print the character
AA  2020             jsr   killtog      ;kill shft/ctrl/comm
KN  2030             rts
OG  2040 ;
AG  2050 kbuf        =     *
KF  2060             ldy   198          ;# chars in buffer
LE  2070             cpy   649          ;max buffer size
NE  2080             bcs   kbuf1        ;buffer full
LK  2090             lda   keys,x       ;get key from table
FJ  2100             sta   631,y        ;put in kbd buffer
AC  2110             inc   198          ;increment buf pointer
GG  2120 kbuf1       =     *
KE  2130             jsr   killtog
IE  2140             rts
MN  2150 ;
GO  2160 ;
AJ  2170 special     =     *            ;handle special chars
BO  2180             cmp   #249         ;stop key
LK  2190             bne   spl1
IC  2200             lda   #$7f
EG  2210             sta   stopflg
IJ  2220             rts
MC  2230 ;
PB  2240 spl1        =     *
LE  2250             sec
FA  2260             sbc   #250         ;must be shift/ctrl/comm
IP  2270             lsr   a
DL  2280             tax
PN  2290             lda   #1
JN  2300             adc   #0
BM  2310             sta   temp2        ;1 if c set, 2 if clr
LD  2320             lda   keyflags,x
AI  2330             eor   temp2        ;switch on/off or
KI  2340             and   temp2        ; lok/off toggle
HJ  2350             sta   keyflags,x
OK  2360 ;
MH  2370 makimag =   *                  ;set up key image
HD  2380             lda   #0
MJ  2390             sta   flagim
LA  2400             ldx   keyflags
IH  2410             beq   nk1
```

```
EF 2420            ora  #1
JM 2430 nk1        =    *
PD 2440            ldx  keyflags+1
DK 2450            beq  nk2
PH 2460            ora  #4
FP 2470 nk2        =    *
JG 2480            ldx  keyflags+2
OM 2490            beq  nk3
FK 2500            ora  #2
BC 2510 nk3        =    *
OB 2520            sta  flagim
OM 2530            rts
CG 2540 ;
MG 2550 ;
EO 2560 killtog    =    *          ;kill 1-time key flags
FF 2570            ldx  #2
OE 2580 kil1       =    *
JE 2590            lda  keyflags,x
HL 2600            and  #2
LJ 2610            sta  keyflags,x
DP 2620            dex
PG 2630            bpl  kil1
NJ 2640            jsr  makimag
GE 2650            rts
KN 2660 ;
EO 2670 ;
IC 2680 drawscrn   =    *          ;draw screenboard
FK 2690            lda  648        ;screen page
IL 2700            sta  screen+1
BI 2710            lda  #0
KA 2720            sta  screen
HN 2730            sta  togcnt
CC 2740            sta  tog2
FA 2750            ldx  #0
DB 2760            ldy  #0
OB 2770 dr1        =    *
DG 2780            lda  #32
GK 2790            sta  (screen),y
AG 2800            lda  keys,x
MM 2810            and  #63
NH 2820            cpy  rvschar    ;check for revers chr
MI 2830            bne  norvs
IL 2840            ora  #128       ;set high bit to rvrs
BO 2850 norvs      =    *
MA 2860            iny
GP 2870            sta  (screen),y
CK 2880            lda  colour
HJ 2890            sta  $d800,y
ED 2900            iny
KD 2910            inx
AE 2920            cpx  #klen
GE 2930            bcc  dr1
CP 2940 ;
KP 2950 ;print bottom two kbd rows
GE 2960            ldx  #0         ;character counter
KO 2970 dr2        =    *
AO 2980            lda  xkeys,x
IJ 2990            cmp  #255       ;toggles
FF 3000            bne  notog
AF 3010            jsr  drawtog    ;print off,on,or lok
PC 3020 notog      =    *
IK 3030            and  #63
LI 3040            sta  temp
DI 3050            lda  rvschar    ;check highlight chr
MH 3060            cmp  #120
EG 3070            bcc  norvs2     ;none to highlight
PL 3080            tya
II 3090            clc
JD 3100            sbc  rvschar    ;highlight
JN 3110            cmp  #3         ; three characters
BG 3120            bcs  norvs2     ; if option
LB 3130            lda  temp       ; is selected
AD 3140            ora  #128
JP 3150            sta  temp
JN 3160 norvs2     =    *
PM 3170            lda  temp
MC 3180            sta  (screen),y
IN 3190            lda  colour
NM 3200            sta  $d800,y
KG 3210            iny
AH 3220            inx
LO 3230            cpx  #xklen     ;# of chars
FL 3240            bmi  dr2
IC 3250 ;
HO 3260 ;underline keyboard
KD 3270            ldx  #40        ;print 40 chars
EC 3280 dr3        =    *
AC 3290            lda  #64        ;underline char
EK 3300            sta  (screen),y
AF 3310            lda  colour
FE 3320            sta  $d800,y
CO 3330            iny
DM 3340            dex
JB 3350            bne  dr3
MA 3360            rts
AK 3370 ;
KK 3380 ;
IK 3390 drawtog    =    *
MP 3400 ;print off, on, or lock message
GA 3410            txa
IN 3420            pha
IK 3430            ldx  togcnt     ;0=shft, 1=ctrl, 2=com
JB 3440            lda  keyflags,x ;off, on, or lok
GG 3450            asl  a
FM 3460            asl  a          ;x4 to point to togtext
EA 3470            clc
KL 3480            adc  tog2
NG 3490            tax
KD 3500            lda  togtext,x
BG 3510            sta  temp
AB 3520            inc  tog2
KP 3530            lda  tog2
KA 3540            cmp  #3
NP 3550            bcc  tgl
HP 3560            inc  togcnt
NN 3570            lda  #0
KG 3580            sta  tog2
NL 3590 tgl        =    *
IJ 3600            pla
FO 3610            tax
BJ 3620            lda  temp
KB 3630            rts
OK 3640 ;
IL 3650 ;
AN 3660 keys       =    *
KG 3670 .asc "abcdefghijklmnopqrs  "
MH 3680 .asc "tuvwxyz0123456789[]  "
BG 3690 .byte 33,34
GB 3700 .asc "#$%&'(),.?:;<=>@↑ "
LB 3710 klen       =    *-keys
PP 3720 ctrlchrs   =    *
CD 3730 .byte 145,17,157,29,148,20,19,147,43,45
FE 3740 .byte 13,249,250,251,252,253,254,255
KM 3750 .asc "*/"
GC 3760 ;
EC 3770 xkeys      =    *
HJ 3780 .asc "  cup cdn clf crt ins del hom clr  +  -  "
LJ 3790 .asc "  rtn stp shf:πππ ctl:πππ com:πππ  *  /  "
MA 3800 xklen      =    *-xkeys
IF 3810 ;
KF 3820 togtext    .asc "off on  lok "
MG 3830 ;
GN 3840 x40        .byte 0,40,80,120,160
AI 3850 ;
JF 3860 keycodes=  *
LG 3870 .byte 10,28,20,18,14,21,26,29,33,34
ML 3880 .byte 37,42,36,39,38,41,62,17,13,60,22,30,31,9
NI 3890 .byte 23,25,12,35,56,59,8,11,16,19,24,27,32
```

Note: The sets of three π symbols in line 3790 are obtained by typing pi's, however, they will appear as 'checkerboards' in lower case mode.

Crystal Ball For The C64 or VIC 20

Ian Adam
Vancouver,
British Columbia

This time of year is one when many sports leagues are very active, and will all too soon come to their seasons' conclusions. Football leads to play offs and championship games, followed by hockey and basketball. These activities quickly become 'media events', with a great deal of coverage, fan interest, office pools, and so on.

We've all seen many computer games based on sports -- those arcade–like games may be realistic simulations of actual sports, or they may be only loosely based on the real thing. I have also seen programs before that will keep track of players' or teams' performance, even tell you which player deserves the game ball on the basis of actual performance. Here's a computer sports application that's very different, however, and it's one that I've never seen described before. This program makes the leap from the game ball to the crystal ball – it will actually "predict" the performance of a number of athletes over a series of future games. Based on a compilation of past player performance, team records, and your prediction of team success, this method is ideally suited to estimating player points over the second half of the season, or to calculating expected point production during play offs.

Applications

The Crystal Ball was originally developed for the hockey play offs, and has been very successful at that application. For our

southern readers, these play offs involve sixteen teams playing up to 89 games in order to decide the winner of the historic Stanley Cup. Predicting the winner of the Cup can be interesting, deciding whether the New York Islanders or Edmonton Oilers will continue whipping the other teams. However, a popular challenge is for a group of fans to get together and try to predict the relative performance of individuals among the 320 players in the play offs. Since the point production of each player may depend on his abilities, the number of games his team stays in the play offs, and the calibre of the opposition, this prediction can be a complex task. This program is ideal for that purpose.

However, it can be equally well applied to a variety of other situations and sports. Most team sports keep track of player production, whether it is hits in baseball, yards gained in football, or dollar value of endorsements. TV networks employ armies of statistical experts to compile and convey this information to the fans. Predicting these statistics can be both enjoyable and challenging; while this program won't do the whole job, it can be a very valuable assistant.

The program itself is straightforward. Much of its length is taken up by making it 'user–friendly', and your effort in typing it in will be repaid many times when using it. It's all in BASIC, so there's nothing too complicated or problematic. After indicating the number of teams and players, RUN the program. All functions are controlled by the main menu. Once you have entered the necessary data, instruct the program to proceed to calculate expected performance, and prepare lists of players ranked by performance and alphabetically. The results are most useful when directed to your printer.

Because of its ample memory, the 64 is ideal for this task. The program is presently configured for analysis of up to 200 players' performance, but could easily be increased to 500 or more. A disk drive makes the task of storing and retrieving data much faster. For the VIC 20, at least 3K of expansion memory would be needed to run the full program, but 8K or more would allow the analysis of more meaningful numbers of players. A stripped–down version is also supplied for the minimum–configuration VIC, modified to use a tape drive instead of disk. With the small number of players you can analyze without memory expansion, the speed of tape storage is not a problem.

Using The Program

So that you can understand the capabilities of the program fully, let's look at the roles of the menu items and subsections:

1. Parameters. Before starting, set the parameters in lines 1220 and 1230 to suit. NT is the exact number of teams you will be examining. M is the maximum number of players to be analyzed and is limited solely by available computer memory.

2. Main Menu. When you RUN the program, you will be presented with the menu. Selections can be made by pressing either a number key or the corresponding function key. If you want to load an existing file from disk or tape, press selection 3 and give the file name. Otherwise, press 1 and start with team data. Choosing any of the other selections at this point would just generate some garbage.

3. Team data. For each team, you must specify:

- a name;
- the number of games of historical data available;
- the team performance over those games (expressed as game points, 2 for a victory, 1 for a tie, 0 for a loss);
- the number of wins and losses expected over the forecast period (the rest of the season, the play offs, or whatever).

The program will prompt for as many teams as you named in line 1230. At any time, you can press RETURN to the name prompt and quit, but you CANNOT then go back and add more teams.

When you have finished entering team data, the computer displays a list of the teams so you can verify the data you entered. Any errors at this stage will be carried forward, so check the data carefully. The computer also adds up total expected wins and losses for all teams. Normally, since each game has a winner and loser, the totals should be the same. If they are different, you may wish to go back and adjust some of your projections. When all is ok, press N for 'no changes', to return to the main menu.

In this segment, the program also calculates a performance factor for each team, based on historic and projected wins and losses (line 1640). A basic assumption is that players will score more points on a team that is winning. A winning score ratio of 5 to 3 is used, which is typical of most sports such as football, hockey, or baseball; for example, a football score might be 28 to 17. One exception to this ratio would be basketball, in which the point spread between winners and losers is typically 10% or less.

4. Player Data. From the main menu, press 2 to enter player data. For each player, you must enter:

- his/her name;
- a team. You may specify a team by its full name, any abbreviation, or its number in the list;
- the number of games for which historical data is available;
- point production during those games. You can use any measure of production, goals, baskets, yards, whatever, as long as it is consistent from one player to another.

You can quit at any time by pressing RETURN when prompted for the name. You will then see a list of all players and their data, one screen full at a time. You will then be asked if there are any changes to be made. If so, press Y, specify the player by number on the list, and enter the revised data as prompted.

Simply press RETURN for any item that does not need to be changed.

If you're getting tired of typing, press N for 'no changes', and return to the main menu. At this point, it is a good idea to save the data to disk or tape. You can come back to this function later to add more players, and the program will remind you where you left off.

5. Calculate and sort. Don't select this option until you have entered the team and player data; also, if you subsequently change any of the data, you will have to come back and repeat this function. This menu selection includes three basic steps:

- each player's projected performance is calculated, based on all the data entered for the player and the team. The results are displayed as calculated.
- the players are ranked in order of maximum production.
- as a cross–index, an alphabetical listing of players and rankings is also produced.

The two rankings are performed by a shell sort in BASIC; although this is very efficient, the sort times do increase with the number of players. If you have many players, take a coffee break. When the two listings are ready, you have the option of viewing or bypassing each one; they go by fast, so use the CONTROL or STOP key if you want to peruse them on the screen.

If any player has fewer than 25 games of historic data, this tends to cast doubt on the statistical validity of the projection because of the small sample size. The program flags this situation beside the player's listing. In football, however, the shorter season would dictate a different threshold, say 10 games. Make the necessary adjustment in line 2430 of the program.

6. Show results. Menu selection 6 allows you to review the rankings without waiting through the calculate and sort steps. Needless to say, the calculating and sorting must have been done previously, or you will get garbage.

7. Print out. Menu selection 7 will send all of the team data and player rankings to your printer. This is the only practical way to review extensive performance listings. A Commodore printer is supported, or any other printer that responds to device #4 on the serial port.

8. Load or Save. Menu selections 3 and 4 provide access to a disk or tape file. All team and player data are saved, but the predictions and rankings are not. After reloading the file using selection 3, you can make any necessary changes to the data, then proceed to calculate, sort, and print the results.

Modifications

The main program listing runs as–is in the Commodore 64, and would require minimal modification for PET/CBM's. It will also run in a VIC with at least 3K of expansion memory if you delete the REM's, but 8K would be preferable. For the VIC, make the changes to lines 1270 and 2900 as shown at the end of the program. The version for the minimum–configuration VIC deletes a lot of the frills such as sound, tidy columns, abbreviations, and function keys. It also has a greatly–reduced capacity for storing team and player data, and assumes tape instead of disk access.

Whatever your sport, I hope you find this program to be a useful and interesting tool. Is the Crystal Ball perfect? Of course not -- there are many intangible factors that can affect a human athlete's performance. One injury can eradicate the most carefully planned prediction. I'm pleased to report, however, that with our input and the Crystal Ball's help, we did win first place in the office pool. May it work as well for you!

Crystal Ball For The C64 and Expanded Vic-20

```
FN  1000 rem save "0:crystal ball 64",8
MD  1010 rem ** the crystal ball for 64 and expanded vic
HO  1020 rem ** written by: ian adam, vancouver, b.c.
GA  1030 cs$=chr$(147): yl$=chr$(30): pk$=chr$(28):
         bk$=chr$(144): bn$=chr$(151)
EH  1040 print cs$: goto 1220
BK  1050 b=1: rem numeric descending sort
DA  1060 b=2*b: if b<np then 1060
AC  1070 b=b/2: if b<1 then return
NK  1080 for i=1 to np-b: c=i
IP  1090 d=c+b: if pl(ix%(c))=>pl(ix%(d)) then 1110
FE  1100 a=ix%(c): ix%(c)=ix%(d): ix%(d)=a: c=c-b:
         if c>. then 1090
CC  1110 next: goto 1070
LK  1120 b=1: rem alphabetic ascending sort
AE  1130 b=2*b: if b<np then 1130
GG  1140 b=b/2: if b<1 then return
DP  1150 for i=1 to np-b: c=i
KG  1160 d=c+b: if pl$(ix%(al%(c)))<=pl$(ix%(al%(d)))
         then 1180
ML  1170 a=al%(c): al%(c)=al%(d): al%(d)=a: c=c-b:
         if c>. then 1160
BG  1180 next: goto 1140
KB  1190 :
GH  1200 : start program
OC  1210 :
AM  1220 dim c,d,i,b,a: m=200: rem max # players
GE  1230 nt=16: rem # teams
ME  1240 :
KL  1250 dim ix%(m),al%(m),pl$(m),pl(m)
FF  1260 dim pl%(m,3),tm$(nt),tm%(nt,3),tm(nt),p$(m)
NJ  1270 w=54276: poke w-3,70: poke w-1,2:
         poke w+2,246: poke w+20,15: poke 53281,1
OD  1280 for i=1 to m: ix%(i)=i: al%(i)=i: next
EN  1290 for i=1 to nt: tm$(i)="   ": next
```

```
II | 1300 :
DF | 1310 : rem menu
MJ | 1320 :
NJ | 1330 print cs$
DF | 1340 print bk$ " [5 spcs]the crystal ball "
   |      tab(45);pk$ " = = = = = = = = = = = = = = = = = " bk$
GH | 1350 print: print "  1. enter team data "
JM | 1360 print: print "  2. enter player data "
AJ | 1370 print: print "  3. load data from disk "
NF | 1380 print: print "  4. save data to disk "
EB | 1390 print: print "  5. calculate and sort "
NM | 1400 print: print "  6. show results "
FP | 1410 print: print "  7. print out results "
AI | 1420 print: print: print "  8. terminate "
IM | 1430 print
HL | 1440 gosub 2890: c = val(r$) + 1
CF | 1450 if asc(r$)>132 then c = 2*(asc(r$)-132) +
   |      7*(asc(r$)>136)
HB | 1460 on c gosub 2820,1510,1940,1690,2220,2390,2470,
   |      2600,2820
CM | 1470 goto 1330
MD | 1480 :
LC | 1490 : enter team data
AF | 1500 :
IB | 1510 print cs$: print " [7 spcs]team data " : if tm%(nt,1)
   |      then 1570
PL | 1520 for t = 1 to nt: a$ = " "
EJ | 1530 print: print: input " enter team name " ;a$
HJ | 1540 gosub 2900: if a$ = " " then 1570
-- | 1550 tm$(t) = left$(a$ + " [9 spcs] " ,11)
FO | 1560 gosub 1620: next
HC | 1570 gosub 3020
IJ | 1580 print: print " change any values  (y/n)? " :
   |      gosub 2890:if r$ = " n " then return
AF | 1590 input " which team " ;a$: gosub 2940: if f then 1600
JC | 1600 gosub 1620: goto 1570
HH | 1610 : get details
KP | 1620 print: input " games played, team pts " ;d,
   |      tm%(t,1):gosub 2900
GC | 1630 print: input " expected wins, losses " ;tm%(t,2),
   |      tm%(t,3): gosub 2900
BD | 1640 tm(t) = d*(3*tm%(t,3) + 5*tm%(t,2))/(3*d + tm%(t,1))
OI | 1650 return
AP | 1660 :
PJ | 1670 : load disk data
EA | 1680 :
ME | 1690 print cs$;yl$: print " [4 spcs]load data from disk "
MN | 1700 if np = 0 then 1730
KC | 1710 print: print: print pk$ " sure you want to lose this data? "
GJ | 1720 gosub 2890: if r$<> " y " then return
HA | 1730 print: print: print pk$ " load which file? " ;
KO | 1740 input df$: gosub 2900
OO | 1750 r$ = " 0: " + df$ + " ,s,r "
GF | 1760 open 15,8,15, " i " : open 8,8,2,r$: gosub 3210:
   |      input#8,a$
JP | 1770 if left$(a$,7) = " crystal "  then 1800
HE | 1780 print cs$;pk$: print " incompatible file " bk$ "  " df$:
   |      print a$
IP | 1790 gosub 2890: close 8: close 15: return
LC | 1800 input#8,a,np
CJ | 1810 if np>m or a>nt then 3270
HC | 1820 nt = a: for i = 1 to nt: input#8,tm$(i),tm(i)
-- | 1830 tm$(i) = left$(tm$(i) + " [11 spcs] " ,11)
IA | 1840 for j = 1 to 3: input#8,tm%(i,j)
EP | 1850 next: next
FL | 1860 for i = 1 to np: input#8,pl$(i)
FB | 1870 for j = 1 to 3: input#8,pl%(i,j)
CB | 1880 next: next
MP | 1890 close 8: gosub 3210: close 15
II | 1900 return
KO | 1910 :
EF | 1920 : enter player data
OP | 1930 :
PM | 1940 print cs$ " [4 spcs]enter player data " : print
DB | 1950 if np = m then 2060
IC | 1960 if np then print " last player:  " pl$(np),tm$(pl%(np,1))
FE | 1970 for i = np + 1 to m
JM | 1980 print: print " player, team name or #, games, points "
PF | 1990 r$ = " " : a$ = " " : print i;
PH | 2000 input r$,a$,pl%(i,2),pl%(i,3): gosub 2900
CG | 2010 if r$ = " " then 2060
GC | 2020 if pl%(i,2) = 0 then print pk$ " i can't handle
   |      that " bk$ " !! " : goto 1980
-- | 2030 pl$(i) = left$(r$ + " [8 spcs] " ,10)
NJ | 2040 gosub 2940: if f then 1980
KG | 2050 pl%(i,1) = t: np = np + 1: next
FB | 2060 gosub 3120
CL | 2070 print: print " change any data  (y/n)? " : print
KB | 2080 gosub 2890: if r$ = " n " then return
KO | 2090 i = 0: print: input " player # " ;i: gosub 2900:
   |      if i = 0 then 2060
PK | 2100 print pl$(i)tm$(pl%(i,1))pl%(i,2),pl%(i,3)
CA | 2110 print: print: print " change data or press return: " : print
NP | 2120 print pl$(i);: input "  or " ;pl$(i): gosub 2900
EH | 2130 a$ = tm$(pl%(i,1)): print a$;: input " or " ;a$
   |      : gosub 2900
CG | 2140 print pl%(i,2);: input " games, or " ;pl%(i,2): gosub 2900
CN | 2150 print pl%(i,3);: input " points, or " ;pl%(i,3): gosub 2900
MB | 2160 pl$(i) = left$(pl$(i) + " [8 spcs] " ,10): gosub 2940:
   |      if f then 2090
NB | 2170 pl%(i,1) = t
OI | 2180 print: print pl$(i)tm$(pl%(i,1))pl%(i,2)pl%(i,3):
   |      goto 2090
CA | 2190 :
PJ | 2200 : save data to disk
GB | 2210 :
FJ | 2220 print cs$;yl$: print " [4 spcs]save data to disk "
LF | 2230 print: print: print " save under what name? " : input df$
AL | 2240 df$ = " 0: " + df$ + " ,s,w
LL | 2250 open 15,8,15, " i " : open 8,8,2,df$
JH | 2260 print#8, " crystal " : gosub 3210
EC | 2270 print#8,nt: print#8,np
KJ | 2280 for i = 1 to nt: print#8,tm$(i): print#8,tm(i)
ON | 2290 for j = 1 to 3: print#8,tm%(i,j)
GL | 2300 next: next
PJ | 2310 for i = 1 to np: print#8,pl$(i)
LO | 2320 for j = 1 to 3: print#8,pl%(i,j)
EN | 2330 next: next
OL | 2340 close 8: gosub 3210: close 15
KE | 2350 return
MK | 2360 :
PN | 2370 : calculate & sort results
```

```
AM | 2380 :
JJ | 2390 if np = 0 then print " enter data first! " : gosub 2890:
       return
CC | 2400 print: print " calculating "
OL | 2410 d = 3: b = 10: a = 2: for i = 1 to np:
       p = int(pl%(i,d)*b*tm(pl%(i,1))/pl%(i,a))
AJ | 2420 pl(i) = p: a$ = right$( " [2 spcs] " + str$(p),4):
       p$(i) = left$(a$,3) + " . " + right$(a$,1)
BE | 2430 if pl%(i,a)<25 then p$(i) = p$(i) + " ** " +
       str$(pl%(i,a)) + " games "
DP | 2440 print i,pl$(i)p$(i): next
AO | 2450 print: print: print: print " sorting. . . ": gosub 1050:
       gosub 1120
BF | 2460 print: print " sorted " : print: gosub 2900
AN | 2470 print " want to see the ranking? " : print: gosub 2890:
       if r$ = " n " then 2510
NC | 2480 print pk$ " rank[3 spcs]player[5 spcs]team
       [6 spcs]production " bk$: print
PF | 2490 for i = 1 to np: a = ix%(i)
PD | 2500 print i tab(7) pl$(a) "   " tm$(pl%(a,1)) " [2 spcs] "
       left$(p$(a),8): next
KG | 2510 print: print: print " want to see the alpha list? "
CN | 2520 gosub 2890: if r$ = " n " thenreturn
HL | 2530 print: print spc(7)pk$ " alphabetical list " bk$: print
AC | 2540 for i = 1 to np: a = al%(i): print a tab(7);
ED | 2550 a = ix%(a): print pl$(a) "   " tm$(pl%(a,1)) " [2 spcs] "
       left$(p$(a),8): next
PA | 2560 gosub 2890: return
OH | 2570 :
MI | 2580 : printout
CJ | 2590 :
LK | 2600 open 4,4: a$ = chr$(10)
LH | 2610 print#4,a$;a$;chr$(14);
GI | 2620 print#4, " ***  crystal ball  *** " chr$(15)
LJ | 2630 print#4,a$;a$;a$;chr$(14) " [3 spcs]teams " chr$(15)
PA | 2640 print#4,a$ " [6 spcs]team[8 spcs]wins[4 spcs]
       losses " a$
CB | 2650 for i = 1 to nt: a = tm%(i,2)
BH | 2660 print#4,right$( " [2 spcs] " + str$(i) + " [3 spcs] " ,6)
       tm$(i) " [2 spcs] " a;spc(5-(a<10))tm%(i,3)
CH | 2670 next
NC | 2680 print#4,a$;chr$(12);chr$(14) " [3 spcs]rankings "
       chr$(15)
FP | 2690 print#4,a$ " rank[2 spcs]player[6 spcs]team[5 spcs]
       production " a$
BD | 2700 for i = 1 to np: a = ix%(i)
PO | 2710 print#4,right$( " [2 spcs] " + str$(i) + " [3 spcs] " ,6)
       pl$(a) " [2 spcs] " tm$(pl%(a,1)) " [3 spcs] " p$(a)
EK | 2720 next
MN | 2730 print#4,a$;chr$(12);chr$(14) " [3 spcs]alphabetical
       list " chr$(15)
HC | 2740 print#4,a$ " rank[2 spcs]player[6 spcs]team[5 spcs]
       production " a$
HD | 2750 for i = 1 to np: a = al%(i)
DG | 2760 print#4,right$( " [2 spcs] " + str$(a) + " [3 spcs] " ,6);
HO | 2770 a = ix%(a): print#4,pl$(a) " [2 spcs] "
       tm$(pl%(a,1)) " [3 spcs] " p$(a)
AG | 2780 next: close 4: return
KF | 2790 :
IG | 2800 :* end
OG | 2810 :
```

```
MA | 2820 print cs$: print " sure you want to lose the data? " : print
MO | 2830 gosub 2890: if r$<> " y " then return
HC | 2840 print " bonne chance! "
CC | 2850 end
AK | 2860 :
AI | 2870 : keyboard & beep
EL | 2880 :
EN | 2890 poke 198,0: wait 198,3: get r$
MH | 2900 poke w,65: for tx = 1 to 80: next: poke w,0: return
CN | 2910 :
BP | 2920 : identify team
GO | 2930 :
CC | 2940 f = 0: a = val(a$): if a then if a< = nt then t = a:
       print tm$(t): return
MG | 2950 a$ = left$(a$,11)
IJ | 2960 for t = 1 to nt: if a$ = left$(tm$(t),len(a$)) then
       print tm$(t): return
DN | 2970 next: print a$;pk$ " . . .i don't recall that team. " bk$
II | 2980 gosub 2900: f = 1: return
CC | 2990 :
MA | 3000 : display team results
GD | 3010 :
AE | 3020 print cs$;pk$ " [3 spcs]team[8 spcs]pts[3 spcs]
       wins " , " losses " bk$: print
LH | 3030 tw = .: tl = .: for i = 1 to nt: print mid$(str$(i) +
       " [2 spcs] " ,2,3)tm$(i);
CG | 3040 print tm%(i,1),tm%(i,2),tm%(i,3)
CD | 3050 tw = tw + tm%(i,2): tl = tl + tm%(i,3): next
PN | 3060 if tw-tl then print pk$
KN | 3070 print: print " total of " tw " wins, " tl " losses. " bn$
EC | 3080 return
GI | 3090 :
GI | 3100 : list players
KJ | 3110 :
MJ | 3120 i = 1
IM | 3130 j = 1: print cs$;pk$ " [4 spcs]player[5 spcs]
       team[6 spcs]games pts " bk$: print
ME | 3140 print mid$(str$(i) + " [3 spcs] " ,2,4)pl$(i) "   "
       tm$(pl%(i,1))pl%(i,2)pl%(i,3);
EF | 3150 j = j + 1: i = i + 1: if i>np then gosub 2900: return
KJ | 3160 if j<23 then 3140
LE | 3170 gosub 2890: goto 3130
AO | 3180 :
IF | 3190 : disk check
EP | 3200 :
HF | 3210 input#15,a,b$
NE | 3220 if a>19 then print a,b$, " error " : close 8: close 15:
       gosub 2890: goto 1330
KL | 3230 return
MB | 3240 :
MM | 3250 : disk file size
AD | 3260 :
FG | 3270 print " disk file " df$ " is too big "
JI | 3280 print " m = " np: print " nt = " a
PF | 3290 print " change lines: "
OO | 3300 close 8: close 15: list 200-210
CG | 3310 :
OH | 3320 vic-20: line 1270  w = 36874: poke w + 4,15
IE | 3330 :    : line 2900 poke w,250: for tx = 1 to 80: next:
                    poke tx,0: return
```

Crystal Ball For The Un-Expanded Vic-20

```
OL | 1000 rem save " 0:crystal ball vic " ,8
NP | 1010 rem ** the crystal ball for the un-expanded vic
HO | 1020 rem ** written by: ian adam, vancouver, b.c.
GA | 1030 cs$ = chr$(147): yl$ = chr$(30): pk$ = chr$(28):
   |       bk$ = chr$(144): bn$ = chr$(151)
CB | 1040 goto 1190
IH | 1050 b = 1
DA | 1060 b = 2*b: if b<np then 1060
LH | 1070 b = b/2: if b<1 then 1120
NK | 1080 for i = 1 to np-b: c = i
GO | 1090 d = c + b: if p(i%(c)) = >p(i%(d)) then 1110
GL | 1100 a = i%(c): i%(c) = i%(d): i%(d) = a: c = c-b:
   |       if c>. then 1090
CC | 1110 next: goto 1070
OL | 1120 b = 1
AE | 1130 b = 2*b: if b<np then 1130
GG | 1140 b = b/2: if b<1 then return
DP | 1150 for i = 1 to np-b: c = i
JO | 1160 d = c + b: if p$(i%(a%(c)))< = p$(i%(a%(d))) then 1180
FJ | 1170 a = a%(c): a%(c) = a%(d): a%(d) = a: c = c-b:
   |       if c>. then 1160
BG | 1180 next: goto 1140
FO | 1190 m = 12: nt = 4: dimi%(m),a%(m),p$(m),p(m),
   |       p%(m,2),t$(nt),t%(nt,2),t(nt)
FB | 1200 for i = 1 to m: i%(i) = i: a%(i) = i: next
KM | 1210 print cs$;bk$ " 1 team data "
IC | 1220 print " 2 player data "
JD | 1230 print " 3 from tape "
CH | 1240 print " 4 to tape "
FK | 1250 print " 5 calc/sort "
LP | 1260 print " 6 print "
JF | 1270 print: print " 7 end "
FO | 1280 gosub 1940: on val(r$) gosub 1290,1490,1400,
   |       1660,1750,1840,2050: goto 1210
FI | 1290 if t%(nt,1) then 1330
DK | 1300 for t = 1 to nt: a$ = " ": print: input " name " ;a$:
   |       if a$ = " " then 1330
HF | 1310 t$(t) = left$(a$ + " [5 spcs] " ,6)
OO | 1320 gosub 1370: next
LF | 1330 gosub 1980
AG | 1340 print: print " changes? " : gosub 1940:
   |       if r$ = " n " then return
CM | 1350 input " team " ;a$: gosub 1950: if f then 1360
CC | 1360 gosub 1370: goto 1330
NF | 1370 print: input " games, pts " ;d,t%(t,1)
CA | 1380 print: input " exp w, l " ;t%(t,2),t%(t,0)
FG | 1390 t(t) = d*(3*t%(t,0) + 5*t%(t,2))/(3*d + t%(t,1)): return
KH | 1400 print cs$;: input " file " ;r$
AB | 1410 open 1,1,0,r$: input#1,a$: if a$ = " cr " then 1430
ON | 1420 print " bad file " a$: gosub 1940: goto 1480
OF | 1430 input#1,a,np: if np>m or a>nt then
   |       print " too big " : goto 1420
AD | 1440 nt = a: for i = 1 to nt: input#1,t$(i),t(i)
CJ | 1450 for j = 0 to 2: input#1,t%(i,j): next: next
ON | 1460 for i = 1 to np: input#1,p$(i)
GJ | 1470 for j = 0 to 2: input#1,p%(i,j): next: next
BB | 1480 close1: return
IE | 1490 if np = m then 1560
PG | 1500 for i = np + 1 to m
CP | 1510 print: print " player, team, games, pts "
IB | 1520 r$ = " ": a$ = " ": input r$,a$,p%(i,2),p%(i,0)
LI | 1530 if r$ = " " then 1560
ND | 1540 p$(i) = left$(r$ + " [4 spcs] " ,6): gosub 1950:
   |       if f then 1510
FH | 1550 p%(i,1) = t: np = np + 1: next
JB | 1560 gosub 2010
EH | 1570 print: print: print " changes? " : gosub 1940:
   |       if r$ = " n " then return
CP | 1580 i = 0: print: input " pl # " ;i: if i = 0 then 1560
JB | 1590 print p$(i)t$(p%(i,1))p%(i,2)p%(i,0)
DH | 1600 print p$(i);: input " or " ;p$(i)
CA | 1610 a$ = t$(p%(i,1)): print a$;: input " or " ;a$
BG | 1620 print p%(i,2);: input " gms, or " ;p%(i,2)
EI | 1630 print p%(i,0);: input " pts, or " ;p%(i,0)
HG | 1640 p$(i) = left$(p$(i) + " [5 spcs] " ,6): gosub 1950:
   |       if f then 1580
OJ | 1650 p%(i,1) = t: print p$(i)t$(t)p%(i,2)p%(i,0): goto 1580
HK | 1660 print: input " file name " ;r$
NG | 1670 open 1,1,1,r$
JC | 1680 print#1, " cr "
EM | 1690 print#1,nt: print#1,np
DK | 1700 for i = 1 to nt: print#1,t$(i): print#1,t(i)
KK | 1710 for j = 0 to 2: print#1,t%(i,j): next: next
KA | 1720 for i = 1 to np: print#1,p$(i)
OK | 1730 for j = 0 to 2: print#1,p%(i,j): next: next
FB | 1740 close 1: return
GA | 1750 b = 10: for i = 1 to np: p(i) = int(p%(i,0)*b*
   |       t(p%(i,1))/p%(i,2))/b
JP | 1760 print i;p$(i)p(i): next: print " sorting " : gosub 1050
CC | 1770 print: print: print " ready " : print: gosub 1940
GC | 1780 for i = 1 to np: a = i%(i)
DK | 1790 print chr$(157);i;p$(a) "   " t$(p%(a,1))p(a): next
GL | 1800 print: print " abc? " : print: gosub 1940
HN | 1810 for i = 1 to np: a = a%(i): print chr$(157);a;
LC | 1820 a = i%(a): print p$(a) "   " t$(p%(a,1))p(a): next
BD | 1830 gosub 1940: return
DL | 1840 open 4,4: a$ = chr$(10)
BB | 1850 print#4,a$;a$;spc(6) " team[3 spcs]wins
   |       [2 spcs]losses " a$
PN | 1860 for i = 1 to nt: print#4,i " [3 spcs] " t$(i) " [2 spcs] "
   |       t%(i,2) " [3 spcs] " t%(i,0): next
OE | 1870 print#4,a$;a$ " rank[2 spcs]player[2 spcs]team
   |       [2 spcs]prod " a$
KI | 1880 for i = 1 to np: a = i%(i)
NO | 1890 print#4,i " [3 spcs] " p$(a) " [2 spcs] " t$(p%(a,1))p(a)
   |       : next
BN | 1900 print#4,a$ " rank[2 spcs]player[3 spcs]team
   |       [3 spcs]prod " a$
FB | 1910 for i = 1 to np: a = a%(i): print#4,a " [3 spcs] " ;
PF | 1920 a = i%(a): print#4,p$(a) " [2 spcs] " t$(p%(a,1)) "
   |       [3 spcs] " p(a)
OA | 1930 next: close 4: return
HG | 1940 wait 198,3: get r$: return
LI | 1950 f = 0: a = val(a$): if a then t = a: print t$(t): return
BH | 1960 for t = 1 to nt: if left$(a$ + " [4 spcs] " ,6) = t$(t)
   |       then return
NC | 1970 next: print a$;pk$ " ?? " bk$: f = 1: return
KN | 1980 print cs$;pk$ " [3 spcs]team[2 spcs]pts win lose " bk$
   |       : print
IC | 1990 for i = 1 to nt: print i;t$(i);
IB | 2000 print t%(i,1)t%(i,2)t%(i,0): next: return
JG | 2010 print cs$;pk$ " [3 spcs]player gms pts team " bk$
MC | 2020 for i = 1 to np
PJ | 2030 print i;p$(i)p%(i,2)p%(i,0)tab(18)left$(t$(p%(i,1)),3)
CO | 2040 next: return
BB | 2050 print " goto 1210 to recoup "
```

Home Control On A VIC 20

Jean Des Rosiers
Montreal, Quebec

". . .turns on lights, controls a cold storage room, gathers temperature data, and keeps watch on my motorcycle parked in the backyard."

With VIC 20s available at such low prices these days, Mr. Des Rosiers' work approaches the ideal real life application. In fact, the BSR Command Console and remote modules will easily cost more than the VIC and Mr. Des Rosiers' hardware. The BSR system is available at Eatons or Radio Shack, and uses the AC lines already inside the walls of your home to send signals to remote modules plugged into any AC outlet. These signals are sent at a much higher frequency than 60 Hz so they won't interfere with the 120 volt AC power. The remote module then transfers power to whatever is connected to it, and voila! Any AC orb can be controlled at any time! And with the program presented here they can be controlled at any time of any day of the week. To top it off, Mr. Des Rosiers has added eight analog to digital inputs so information can be collected that can be used to determine controller output. The input could come from a simple switch or even a temperature sensor, for which detailed schematics are included! As if that weren't enough, Des Rosiers has also built remote status indicators from 7 segment displays so you can eliminate your TV or monitor and use it elsewhere! M.Ed

I started playing with micro processors when I bought a used KIM. Then I bought the kit version of the Sinclair ZX80. A few years later I got hold of a real micro computer – a DEC VT–180, but when my young daughter always wanted to hammer away at my keyboard, I decided to get her a VIC 20. The ads promised great educational software, plus they had started to drop in price.

The "great educational software" came in expensive and impractical cartridges, and all in english. A french speaking two year old can't be taught the subtleties of the english language in a short enough time to become interrested, and on a VIC 20. So the poor vic was left alone in its box. I later bought a 64 and she quickly learned numbers and the alphabet.

I then tried to sell the poor VIC 20, but nobody was foolish enough to consider buying it. Then reading the 64's programmers reference manual I noticed it had a 24 hour clock, I then checked the VIC 20 to find it had a clock too. That led me to think about the times I wanted to have a computer run a few things around the house but rejected the idea as being too complicated.

I had read the numerous articles that were written throughout the years about home control, most of them either required a lot of hardware modifications or tying up an expensive micro computer, with programs written in hard to adapt machine language. The VIC 20 had none of these limitations. It's cheap, has a clock, a user port, an expansion connector and good basic.

As I started to write the control program it became obvious that the basic amount of memory was not great enough to store the arrays needed by the program, so the first order of business was to build an 8K memory expansion.

An analog to digital converter and an analog multiplexer, giving eight analog measuring points, were added to the same module and hooked up to the expansion connector.

Then I built the interface to hook–up the BSR controller to the user port. A home built battery backed–up power supply made the whole system immune from power outages. I also added a remote display to keep track of what was going on without having the T.V. set on all the time. Most of the hardware described can be built as required. If the full possibilities of the system are not needed, just build whatever interfaces are necessary to make it functional.

Hardware and Software Description

Memory Expansion

The 8K expansion is simply an 8K by 8 bit chip connected on the expansion connector, as shown in figure 1.

BSR Interface

The BSR interface, figure 2, is a modified (to make it work) version of a circuit that appeared in the January 1982 issue of BYTE. Voltage to power the oscillator is provided by the BSR command console itself. Locate a large (1000 uF) capacitor in the command console, connect the respective + and – leads from the interface to the capacitor leads. There should be about 18 volts on that filter cap, but be careful when measuring voltages in the command console because it is not isolated from the A.C. line. That is why an opto–isolator is used to connect the interface to the user port. The output from the oscillator is connected to pin 7 (seven) of the 542C I.C. (the 542C is the only chip used in the BSR consoles). Any command console can be used provided it has a 542C chip. Usually on the mini consoles and the large ones lacking ultrasonic capabilities, pin 7 is grounded. Cut the foil trace leading to pin 7 and connect it to the output of the oscillator in the interface. I added a 15 volts zener to keep the voltage on the 4001 CMOS I.C. from reaching destructive levels. It seems that the old BSR consoles had an 18 volts zener, but the newer ones don't. So to keep the circuit operational it is better to put in the 15 volts zener.

The oscillator output should be 40 Khz with the values given. The frequency doesn't have to be spot on 40 Khz – the BSR will accept commands with frequencies ranging from 33 to 50 Khz, although the operation will be marginal at the extremes of the range.

The software to make this interface mimic the cordless controller works as follows. First, the code to be sent is stored as a variable, eg: the 'ALL ON' command is equal to "0001111100", the pro-

gram then searches this string and pokes the appropriate values in memory. For a zero, the program pokes the values 24 and 136 in two consecutive memory locations. For a one, the numbers poked are 80 and 80. These numbers were chosen to give the correct timing using instruction loops in machine language. When the whole string has been examined and the appropriate values poked, the machine language subroutine takes over and toggles the I/O port according to the values previously poked in memory. The net result is a complete emulation of the BSR command console. Thus the sequence is to first send a unit code, then the action to be taken, just as if someone was pushing the buttons.

A/D Converter

The A/D used is a single channel ADC-0804. A 4051 CMOS analog multiplexer expands it to a total of 8 analog inputs. The analog channel is selected by an octal latch. The I/O port could be used to do the same thing, but would be less elegant. The whole circuit is shown in figure 3. Since 3 decoded addresses were needed (the octal latch and the A/D are memory mapped) and three 8K address blocks were left unused (block 1 is used by the 8K expander), there was no need for an address decoder. So block 2 is used to select an analog channel, block 3 starts an analog conversion and block 5 reads the result from the A/D.

The software used is quite simple. First the selected A/D channel to be used, from 0 to 7, is poked where the machine language subroutine is located, then this value is stored to location 16384 (4000 hex or block 2). Since the octal latch (74LS373) is selected by block 2, we end up with an analog channel being selected. Second, to start a conversion we only need to send a pulse to location 24576 (6000 hex or block 3). To give the A/D time to complete it's conversion, a small delay loop is executed. Then the value from location 40960 (A000 or block 5) is loaded in the accumulator and stored in memory location 16156, which is where the basic part of the program retrieves the result of the conversion. This whole process is repeated 40 times to iron out peaks or stray values.

Temperature Sensors and Amplifiers

I set out to find cheap, easy-to-get and reliable sensors. I finally opted for regular 2N-2222A transistors. The emitter base junction of a silicon transistor will measure about .7 volts when forward biased, and goes down as temperature increases. The change in voltage is minimal, about 2.16 mV/°C or 216 mV from 0°C to 100°C. Since the temperatures I was working with would give me the same range (-50°C to +50°C), I was forced to amplify the signal. As a side benefit, the signal is inverted so that voltage goes up as temperature goes up. The amplifiers are implemented with a pair of LM324's quad op-amps. Now the rate of change is a more measurable 21.6 mV/°C.

The distance from the sensors does not affect the readings, but as wire length increases, so does noise pick-up. When the sensors are to be located more than 10 metres from the amplifiers, put a small (0.1 uF) capacitor at the amplifier input to shunt the noise to ground.

Power Supply

There are two flavours of the VIC 20. One has only 9 volts A.C. input, with the rectifier and the regulator inside the keyboard, and the other has 9 volts A.C. input **and** +5 volts D.C. input (the rectifier, filter capacitor and regulator are inside the power pack as in the 64's). The power supply shown in figure 5 will accommodate both types.

The power supply in figure 5, works as follows. First the 9 volts A.C. from the transformer is rectified and connected to a 12 volt battery. I used a transformer from the early VIC's, which I got from a local parts store, and with it the charge current is a safe 300 mA. The battery can be any size 12 volts. I used a sealed 12 volt industrial lead-acid battery, but a small motorcycle battery or the right number of nickel-cadmium cells would do as well. The 12 volts from the battery is brought down to 5 volts using a 4 ohm resistor feeding an LM-323 three amp regulator. The resistor is used to reduce the input voltage to the LM-323, otherwise it's temperature would climb too high and it would cause a thermal shut-down. Even though the input voltage is reduced to 8 volts, it is still wise to heat sink the regulator. The 9 volts A.C. is fed to the VIC as usual. In the event of a loss of power the battery/+5 volts regulator will keep the VIC 20 humming. The only thing not working would be the cassette interface.

Another way to get the same backup would be to use alkaline "D" cells with rectifiers used as current steering diodes. For the old style VIC's use 6 "D" cells to get 9 volts and hook this up to the filter cap preceding the 5 volts regulator (see figure 6). On the newer VIC's use 4 cells with 2 rectifiers and tap this last combo in the power cable, as in figure 7.

Remote Display

Figure 8 is a schematic to giving a visual indication of what is going on, without having to keep the TV or monitor on. On my system I used two displays of four digits each. One of the displays is used to show the time, and is updated each time the program goes through the main loop. The other is used to show the temperature of one of the sensors, chosen by keyboard entry. The method of transmission between the VIC-20 and the displays is serial, with a small peculiarity; the same wire that sends the data is used to power the display. Only three wires are used to connect the VIC to the two displays, so that it can be located quite far without having a mess of wires strung all over the place or having to hunt for an outlet to power the displays.

A/D Calibration

It is not necessary to trim every channel so that they respond the same way. The required offset will be done by software. First measure the voltage at the output of the LM324 amplifier with a sensor immersed in crushed ice, then take a reading with the sensor plunged in boiling water, this will give you the range for 100°C, on my system the range was 1.6 volts at 0°C and 3.76 volts at 100°C, thus:

$$(3.76-1.6)/100=0.0216\text{v/°C}$$

Since a temperature of 0°C gives around 1.6 volts at the output of the LM324 amplifiers (on my system), we can extrapolate that at -50°C the output would be 1.6-(50 X .0216) or .52 volts. Lets set it to .5 volts, to give a bit of leeway, this last value will be the voltage required on pin 7 of the ADC-0804. Turn the proper pot until the voltage on pin 7 is equal to .5 volts. To set the voltage on pin 9 (VREF/2), the calculation is as follows. The range is 100°C (-50°C to +50°C), so 100 X .0216 = 2.16 volts and VREF/2 = 2.16/2 = 1.08 volts. Lets set it to 1.1 volts. Turn the other pot until the voltage measured on pin 9 is equal to 1.1 volts.

If absolute precision is not required, the potentiometers can be replaced by a resistor divider network as shown on the diagram.

Now connect all the sensors to the amplifiers, and the amplifiers to the A/D, and run the program in listing 2. Plunge the sensor to be

calibrated in a glass filled with a mixture of ice and water. Let the reading stabilize and note the reading (The first reading is for C%(0) and the last C%(7) giving all eight channels). This is the offset to be used with that channel. Note the readings for all eight sensors (if they are all used) and insert the proper values in the main program.

Listing 2 : Temperature Sensor Offset

```
10 print "■" : for i=1 to 7
20 gt=0                          GT is Grand Total
30 poke 16384,i                  select analog channel
40 for j=0 to 19                 read sensor 20 times
50 poke 24576,0                  start conversion
60 gt=gt+peek(40960)             read result and add to previous
70 next j
80 result=127-(gt/20)            result is offset
90 print int((int(result*100))/100)   drop all fractions
100 next i
110 for i=0 to 999:next i        1 second delay
120 goto 10
```

The comments tell the story pretty well. Using the offset generated by this small routine, I have tested two sensors one next to the other, and have found them to be accurate to within 0.5°C from 0°C to 45°C. The temperature readings given by both sensors were exactly the same throughout the range.

Program Structure: Listing 1

Line(s)	Description
100	256 bytes are reserved for the machine language subroutines from 16128 to 16383.
110	The offset required by the temperature sensors are inserted on this line.
120	Arrays are DIMed here, OF is clock offset.
130 to 220	BSR message formats
230 to 370	Data statements for machine language routines
360 to 460	Constants and set the main array to a known value.
470 to 590	Menu display.
600 to 660	Set clock and day.
670 to 900	Set "Action array" routines.
910 to 980	Set BSR modules NOW.
990 to 1020	Load array from tape.
1030 to 1060	Save array to tape.
1070	Get character from keyboard, if equal to **T**erminate, go back to menu.
1080 to 1100	If character input is from 1 to 8, send value read from sensor selected to remote display.
1110 to 1130	Search array for a match to the present time and set the appropriate BSR module on or off. Also send the time to the remote display.
1140 to 1160	Go read temperature and change day at midnight.
1180 to 1240	Every ten minutes reset BSR modules as they should be, in case of a power outage.
1250	End main loop.
1260 to 1290	Search a secondary array to make sure BSR modules are not toggled twice, and go do it if it is not done.
1300 to 1320	Send the proper module number and the proper action to be taken.
1330 to 1350	Set all sixteen modules from the NOW command.
1360 to 1390	Scan the BSR command string and poke the proper values in memory.
1400 to 1450	Read all eight temperature sensors and display results on the screen as well as the time and day of the week.
1460 to 1490	Send data to the remote displays.
1500 to 1550	Clock offset routine (see text).

Temperature and time related decisions can be done between lines 1090 and 1100. The values for all eight temperature sensors are contained in AC(0) to AC(7), and are in °C. The user port can be used as output to sound a siren for an alarm or as inputs to read alarm sensors. If more bits of the user port are used as output don't forget to change address 16161 in the BSR interface driver to reflect what bits are used as output. In my program I used PB0 to PB2 as outputs. PB0 is connected to the BSR interface, PB1 and PB2 are used for the remote displays.

I had to include a "Clock offset routine" because the 24 hour clock on the VIC 20 is slightly fast, about 50 seconds a day fast. There were two ways to go about this, first make a hardware crystal oscillator or second make the software trim the clock. The routine included will trim 2 seconds per hour so the VIC ends up keeping the time almost perfectly.

Since this is not a commercial endeavour it is possible to enter wrong data, and the program will do funny things. I trust that someone smart enough to duplicate all or part of this package would not be foolish enough to enter wrong values.

Explanation Of Variables

C%(x)	Temperature offset. Range of x,0 to 7.
C$(x)	BSR message string. Range of x,1 to 16.
T%(I,J,A)	Main action array. A number is stored that represents the time and action to be taken in the following format HHMMA where HHMM is hours and minutes. A is the action, a 1 means turn on at the specified hour and a zero, off. The number stored can't be higher than 23591 or lower than 0. Range I to 15, J to 6 and A to 3. I represents channel number, J is the day of the week and A is one of four actions to be taken per day.
X%(I,A)	Daily array, a 1 means the action was taken. Needed to keep from sending a BSR message on every loop of the program. Range I to 15 and A to 3.
K1	A constant equal to 0.3921568
JO$(x)	Days of the week. Range of x,0 to 6.
E	Day of the week, 0=monday, 1=tuesday and so on.
CA	Value of character typed on keyboard. Used to send a temperature value to the remote display.
CB	Equals CA-1
HA	Value to be poked in 16157 for the remote display. Represents the thousandths and hundreds.
HB	Value to be poked in 16158 for the remote display. Represents the tens and units.
HC	Value to be poked in 16159 for the remote display. Used to select user port bit 2 or 4.
AC(x)	Value in °C of the temperature read for that sensor range of x,0 to 7
H$	Four leftmost characters read from the VIC's clock.
HH$	Middle two characters read from the clock. Used to reset BSR modules every 10 minutes.
HE%	Value of H$
SW	Switch to keep from incrementing the days counter more than once at midnight. Value 0 or 1.
S1	Switch to keep from sending a BSR message more than once every ten minutes. Value 0 or 1.
TA%	HHMMA transferred from T%(I,J,A)
TB%	HHMM extracted from TA%
AC%	A extracted from TA%
EN$	BSR message string to be sent

GT	Total of 40 sensor values
A	Average of the 40 values (=GT/40)
Y	Temperature value with offset added
C	Y to two places. (°CC.CC)

Machine Language Routines

BSR Interface

```
16160  169,1        lda #7
16162  160,0        ldy #0
16164  141,18,145   sta 37138  (DDR for port B)
16167  169,1        lda #1
16169  141,16,145   sta 37136  (DATA reg. for port B)
16172  32,69,63     jsr 16197
16175  234          nop
16176  234          nop
16177  234          nop
16178  169,0        lda #0
16180  200          iny
16181  141,16,145   sta 37136  (DATA reg. for port B)
16184  32,69,63     jsr 16197
16187  234          nop
16188  234          nop
16189  234          nop
16190  200          iny
16191  192,24       cpy #24
16193  208,228      bne 16167
16195  96           rts
16196  234          nop
16197  169,0        lda #0
16199  105,1        adc #1
16201  32,81,63     jsr 16209
16204  201,11       cmp #11
16206  48,247       bmi 16199
16208  96           rts
16209  190,0,63     ldx 16128,Y
16212  202          dex
16213  208,253      bne 16212
16215  96           rts
```

The first three lines set up the user port to a known state. Then the program jumps to a couple of instruction loops that provide the proper delay. Line 16209 loads the X register with the values that were poked in memory by the basic part of the program, since X is loaded from address 16128 indexed by Y, we manage to scan the 24 memory locations containing the values chosen to give the proper timing relationship. Since the values chosen gave only one tenth of the time required, lines 16199 to 16206 make the subroutine starting from 16209 repeat ten times.

A/D Converter

```
16216  169,0        lda #0      (Channel # in 16217)
16218  141,0,64     sta 16384   (Octal latch)
16221  141,0,96     sta 24576   (Starts conversion)
16224  162,85       ldx #85     (delay)
16226  202          dex
16227  208,253      bne 16226
16229  173,0,160    lda 40960   (A/D chip)
16232  141,28,63    sta 16156
16335  96           rts
```

Basic pokes the channel number in memory location 16217, this value is stored into the octal latch thus selecting an analog channel. Then a conversion is started by line 16221. A small instruction loop at 16224 gives the A/D time to finish it's conversion. The A/D converter is then read in line 16229 and the result stored in memory location 16156. The value in this last location is retrieved and treated by the basic part of the program.

Remote Display

```
16236  162,100      ldx #100    (To repeat 100 times)
16238  172,29,63    ldy 16157   (Thousands + Hundreds)
16241  173,31,63    lda 16159   (Port bit used)
16244  141,16,145   sta 37136   (Data reg. for port B)
16247  169,0        lda #0
16249  141,16,145   sta 37136   (Data reg. for port B)
16252  136          dey
16253  208,242      bne 16241
16255  202          dex
16256  208,236      bne 16238
16258  96           rts
16259  172,30,63    ldy 16158   (Tens + units)
16262  173,31,63    lda 16159   (Port bit used)
16265  141,16,145   sta 37136   (Data reg. for port B)
16268  169,0        lda #0
16270  141,16,145   sta 37136   (Data reg. for port B)
16273  136          dey
16274  208,242      bne 16262
16276  96           rts
```

This subroutine is used to send data to the displays. The interface to the basic part of the program is done through memory locations 16157 to 16159. The first two locations contain the thousands, hundreds, and the tens units of the number to be transmitted, and the last contains the bit value of the port used. The program consists of instruction loops controlled by the values in memory. The two routines (16236 to 16258 and 16259 to 16276) are basically the same, except that the first one is repeated 100 times to give the correct number of times that the port bit is toggled. Any number up to 9999 can be sent, as long as the proper values are poked in memory.

BSR Command Codes

D0	D1	D2	D3	D4	Function	String Name
0	0	0	1	1	ALL ON	TA$
0	0	0	0	1	ALL OFF	TE$
0	0	1	0	1	ON	A$
0	0	1	1	1	OFF	E$
0	1	0	1	1	BRIGHTEN	NOT DEFINED
0	1	0	0	1	DIM	D$
0	1	1	0	0	CHANNEL 1	C$(1)
1	1	1	0	0	CHANNEL 2	C$(2)
0	0	1	0	0	CHANNEL 3	C$(3)
1	0	1	0	0	CHANNEL 4	C$(4)
0	0	0	1	0	CHANNEL 5	C$(5)
1	0	0	1	0	CHANNEL 6	C$(6)
0	1	0	1	0	CHANNEL 7	C$(7)
1	1	0	1	0	CHANNEL 8	C$(8)
0	1	1	1	0	CHANNEL 9	C$(9)
1	1	1	1	0	CHANNEL 10	C$(10)
0	0	1	1	0	CHANNEL 11	C$(11)
1	0	1	1	0	CHANNEL 12	C$(12)
0	0	0	0	0	CHANNEL 13	C$(13)
1	0	0	0	0	CHANNEL 14	C$(14)
0	1	0	0	0	CHANNEL 15	C$(15)
1	1	0	0	0	CHANNEL 16	C$(16)

Listing 1: Home Control Program

```
OL  100 poke56,63:clr
--  110 c%(0)=17:c%(1)=21:c%(2)=19:c%(3)=18
        :c%(4)=27:c%(5)=0:c%(6)=0:c%(7)=0
CH  120 dimc$(16):dimt%(15,6,3):dimtt%(15):dimx%(15,3)
        :k1=100/255:poke36864,6:of=2
LJ  130 c$(1)="0110010011":c$(2)="1110000011"
BL  140 c$(3)="0010011011":c$(4)="1010001011"
JM  150 c$(5)="0001011101":c$(6)="1001001101"
LN  160 c$(7)="0101010101":c$(8)="1101000101"
JL  170 c$(9)="0111010001":c$(10)="1111000001"
BJ  180 c$(11)="0011011001":c$(12)="1011001001"
PJ  190 c$(13)="0000011111":c$(14)="1000001111"
GH  200 c$(15)="0100010111":c$(16)="11000001"
BF  210 a$="0010111010":e$="0011111000"
BA  220 ta$="0001111100":te$="0000111110"
        :d$="010011
DB  230 data 169,  7,160,  0,141, 18,145,169
NA  240 data   1,141, 16,145, 32, 69, 63,234
JE  250 data 234,234,169,  0,200,141, 16,145
AH  260 data  32, 69, 63,234,234,234,200,192
MF  270 data  24,208,228, 96,234,169,  0,105
KJ  280 data   1, 32, 81, 63,201, 11, 48,247
FF  290 data  96,190,  0, 63,202,208,253, 96
MF  300 data 169,  0,141,  0, 64,141,  0, 96
JP  310 data 162, 85,202,208,253,173,  0,160
AF  320 data 141, 28, 63, 96,162,100,172, 29
GH  330 data  63,173, 31, 63,141, 16,145,169
AJ  340 data   0,141, 16,145,136,208,242,202
MI  350 data 208,236, 96,172, 30, 63,173, 31
JG  360 data  63,141, 16,145,169,  0,141, 16
OK  370 data 145,136,208,242, 96
LK  380 jo$(0)="monday":jo$(1)="tuesday"
NJ  390 jo$(2)="wednesday":jo$(3)="thursday"
GP  400 jo$(4)="friday":jo$(5)="saturday"
KP  410 jo$(6)="sunday":sw=0:s1=0
DA  420 fori=0to15:forj=0to6:fork=0to3
PB  430 t%(i,j,k)=12000:next:next:next
DF  440 for i=16160 to 16276:reada:pokei,a:next
DL  450 poke16128,80:poke16129,80:poke16150,240
FF  460 poke16151,240:poke37138,7:poke37136,6
        :poke37136,0
KI  470 print"{rvs} S {off}":print"set {rvs} c {off}lock"
LD  480 print"set {rvs} a {off}ction arrays":print"{rvs} r {off}un
FH  490 print"{rvs} l {off}oad array":print"{rvs} s {off}ave array"
NP  500 print"{rvs} e {off}xit":print"set bsr {rvs} n {off}ow"
GP  510 inputr$:ifr$="c"thengoto600
ME  520 ifr$="a"thengoto670
II  530 ifr$="l"thengoto990
OM  540 ifr$="s"thengoto1030
LH  550 ifr$="e"thengoto590
MI  560 ifr$="n"thengoto910
LA  570 ifr$<>"r"thengoto470
ME  580 gosub1070:goto470
OE  590 end
II  600 print"{rvs} S {off}":printti$,jo$(e):poke214,10
BC  610 print:print"n=no change":poke214,4
KO  620 print:input"time";t$:ift$="n"thengoto650
PE  630 ti$=t$
AG  640 input"day  0=monday. . .";e
        :if(e>6ore<0)thengoto640
JF  650 printti$,jo$(e)
CE  660 fori=1to2000:next:goto470
JL  670 print"{rvs} S {off}":poke214,13
DK  680 print:print"n=no change":print"a=next action"
PJ  690 print"e=exit":print"d=next day"
        :print"m=next module
CA  700 print"time  hhmma a=0 off"
IE  710 print"[12 spcs]a=1 on"
GH  720 forj=0to6:poke214,0:print
FD  730 print"[14 spcs]";
PA  740 poke214,0:print:printjo$(j):print
AE  750 fora=0to3:poke214,2:print:print"time+action";a+1
MH  760 fori=0to15
BK  770 poke214,9:print:print"[12 spcs]";
EN  780 poke214,9:print:printt%(i,j,a):poke214,6
CJ  790 print:print"module[15 spcs]";
BG  800 poke214,6:print:print"module";i+1:poke214,7
FG  810 print:print"[10 spcs]";:poke214,7
PL  820 print:inputre$:ifre$="n"thengoto880
CN  830 ifre$="e"thengoto470
FO  840 ifre$="d"thengoto900
HB  850 ifre$="m"thengoto880
CP  860 ifre$="a"thengoto890
NF  870 t%(i,j,a)=val(re$)
PE  880 nexti
BE  890 nexta
EK  900 nextj:goto470
JM  910 poke214,13:print:print"0=off":print"1=on"
NL  920 fori=0to15:poke214,10
ME  930 print:print"[23 spcs]"
NB  940 poke214,10:print:print"module";i+1
DF  950 inputre:ifre=0thentt%(i)=re:goto980
BB  960 ifre=1thentt%(i)=re:goto980
DI  970 goto930
OJ  980 next:gosub1330:goto470
HI  990 open6,1,0
PI  1000 forj=0to6:fora=0to3:fori=0to15
OI  1010 input#6,v%:t%(i,j,a)=v%:next:next:next
MB  1020 close6:goto470
HK  1030 open6,1,2,"data"
HL  1040 forj=0to6:fora=0to3:fori=0to15
FK  1050 print#6,t%(i,j,a):next:next:next
EE  1060 close6:goto470
PK  1070 getc$:ifc$="t"thenreturn
CH  1080 ca=val(c$):ifca=1orca=2orca=3orca=4orca=5
        orca=6orca=7orca=8thencb=ca-1
LF  1090 ha=(cb+1)*10:ifac(cb)<0thenha=((cb+1)*10)+2
LM  1100 hb=abs(int(ac(cb))):hc=4:gosub1460
BN  1110 h$=left$(ti$,4):hh$=mid$(ti$,3,2):he%=val(h$)
KI  1120 he%=val(h$):ha=int(he%/100):hb=he%-(ha*10
CP  1130 hb=he%-(ha*100):hc=2
HK  1140 gosub1460:gosub1260:gosub1400
KL  1150 if(val(h$)=0andsw=0)thensw=1:e=e+1
        :ife=7thene=0
FI  1160 if(val(h$)<>0)thensw=0
EP  1170 hs=val(right$(ti$,2))
CD  1180 if(hh$="00"ands1=0andhs>20)thens1=1
        :gosub1330:gosub1500:goto1070
AB  1190 if(hh$="10"ands1=0)thens1=1:gosub1330:goto1070
LB  1200 if(hh$="20"ands1=0)thens1=1:gosub1330:goto1070
GC  1210 if(hh$="30"ands1=0)thens1=1:gosub1330:goto1070
BD  1220 if(hh$="40"ands1=0)thens1=1:gosub1330:goto1070
MD  1230 if(hh$="50"ands1=0)thens1=1:gosub1330:goto1070
GD  1240 if(mid$(ti$,4,1)="1")thens1=0
ON  1250 goto1070
GO  1260 fori=0to15:fora=0to3:ta%=t%(i,e,a):tb%=int(ta%/10)
        :ac%=ta%-(tb%*10)
BA  1270 if(he%=tb%andx%(i,a)=0)thentt%(i)=ac%
        :gosub1300:x%(i,a)=1
NI  1280 if(he%<>tb%andx%(i,a)=1)thenx%(i,a)=0
```

```
KB | 1290 next:next:return
CL | 1300 en$ = c$(i + 1):gosub1360:ifac% = 1thenen$ = a$
KF | 1310 ifac% = 0thenen$ = e$
NB | 1320 gosub1360:return
OP | 1330 fori = 0to15:en$ = c$(i + 1):gosub1360:iftt%(i) = 0
   |         thenen$ = e$
DI | 1340 iftt%(i) = 1thenen$ = a$
BG | 1350 gosub1360:next:return
DI | 1360 forj = 0to9:c$ = mid$(en$,j + 1,1)
AK | 1370 ifc$ = "0" thenpoke16130 + j*2,24:poke16131 + j*2,136
OH | 1380 ifc$ = "1" thenpoke16130 + j*2,80:poke16131 + j*2,80
OO | 1390 next:sys16160:return
FN | 1400 print "S" :printti$,jo$(e):print
CJ | 1410 fori = 0to7:poke16217,i:gt = 0
AE | 1420 forj = 0to39:sys16216
AC | 1430 a = peek(16156):gt = gt + a:next
IP | 1440 a = int(gt/40):y = (k1*(a + c%(i)))-50:c = (int(y*100))/100
BL | 1450 print "sensor" ;i + 1,c:ac(i) = c:next:return
OI | 1460 poke16157,ha:poke16158,hb:poke16159,hc
   |         :poke37136,hc:poke37136,0
MO | 1470 fork = 1to1:next:ifha>0thensys16236
MH | 1480 ifhb>0thensys16259
OO | 1490 return
HK | 1500 hd = val(ti$):hd = hd-of:ifhd<10thenti$ = "00000" +
   |         right$(str$(hd),1):return
BF | 1510 ifhd<100thenti$ = "0000" + right$(str$(hd),2):return
IK | 1520 ifhd<1000thenti$ = "000" + right$(str$(hd),3):return
LG | 1530 ifhd<10000thenti$ = "00" + right$(str$(hd),4):return
KD | 1540 ifhd<100000thenti$ = "0" + right$(str$(hd),5):return
KO | 1550 ifhd>95959thenti$ = right$(str$(hd),6):return
```

List Of Figures

Figure 1 Single Chip 8K Memory Expansion
Figure 2 BSR Interface
Figure 3 Analog to Digital Converter
Figure 4 Temperature Sensors and Amplifiers
Figure 5 Power Supply for both style VIC 20's
Figure 6 Alkaline Cells Battery Back-up (old VIC 20's)
Figure 7 Alkaline Cells Battery Back-up (new VIC 20's)
Figure 8a Remote Display Transmitter
Figure 8b Remote Display Receiver

References

Article	Author	Publication	Date
Computerize A Home	Steve Ciarcia	BYTE	Jan 80
Plug In Remote Control System	Steve Ciarcia	Radio Electronics	Sep 80
An 8080–Based Remote Appliance Controller	David Stoehlin	BYTE	Jan 82
Single Wire Pair Multiplexes Power and Data For Display	Tommy N. Tyler	Electronics	Dec 78

Parts List

Semiconductors

1	HM6264P-15	8k X 8 Static RAM chip
1	TIL-111	opto coupler
3	2N-3904	NPN Transistor
1	1N-4744A	15 Volt 1 Watt Zener Diode
1	4001	Quad 2-Input NOR Gate
8	2N-2222A	NPN Transistors
2	LM-324	Quad OP-AMP
4	4033	Decade counter with 7 segment output
4	MAN-3	7 segment display
1	1N-4001	Rectifier
1	74LS05	Hex inverter with open collector
1	TIP-117	Darlington PNP Power Transistor
1	74LS04	Hex inverter
1	74LS00	Quad 2-input NAND Gate
1	4051	8 Channel Multiplexer
1	74LS373	Octal Latch
1	ADC-0804	8 Bit A/D Converter
1	LM-323	3 Amp 5 VOlt Regulator
1	KBPC25-02	25 Amp Bridge Rectifier
3		6 Amp Rectifier

Resistors

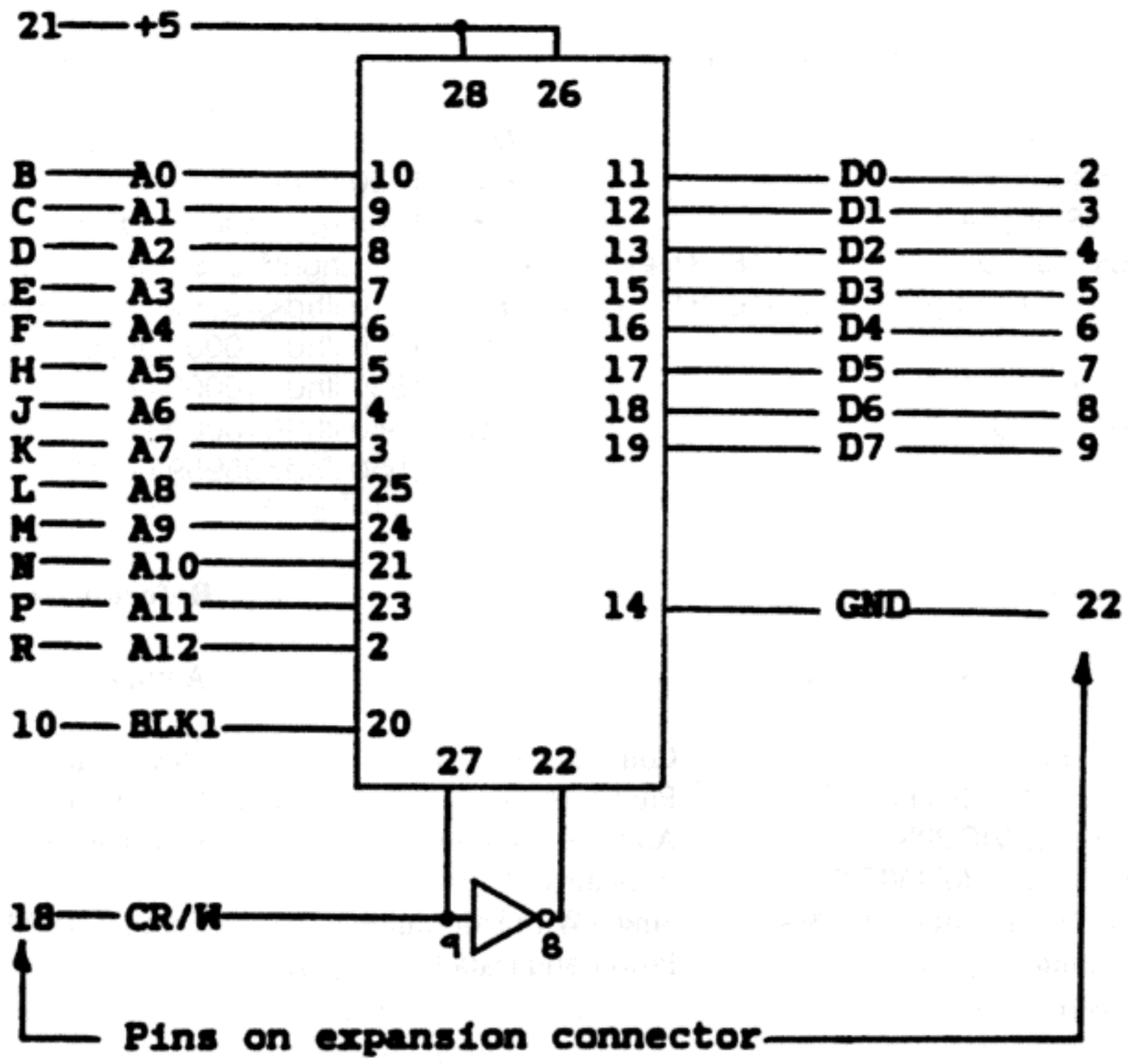
1	4	25 W
1	30	1/4 W
1	100	1/4 W
1	120	1/4 W
1	180	1/4 W
2	750	1/4 W
2	1k	1/4 W
1	5.1k	1/4 W
11	9.1k	1/4 W
3	10k	1/4 W
2	33k	1/4 W
8	100k	1/4 W
8	1M	1/4 W

Capacitors

1	200 pf
1	470pf
1	0.05 uf
1	0.33 uf
1	100 uf
1	500 uf

FIGURE 1

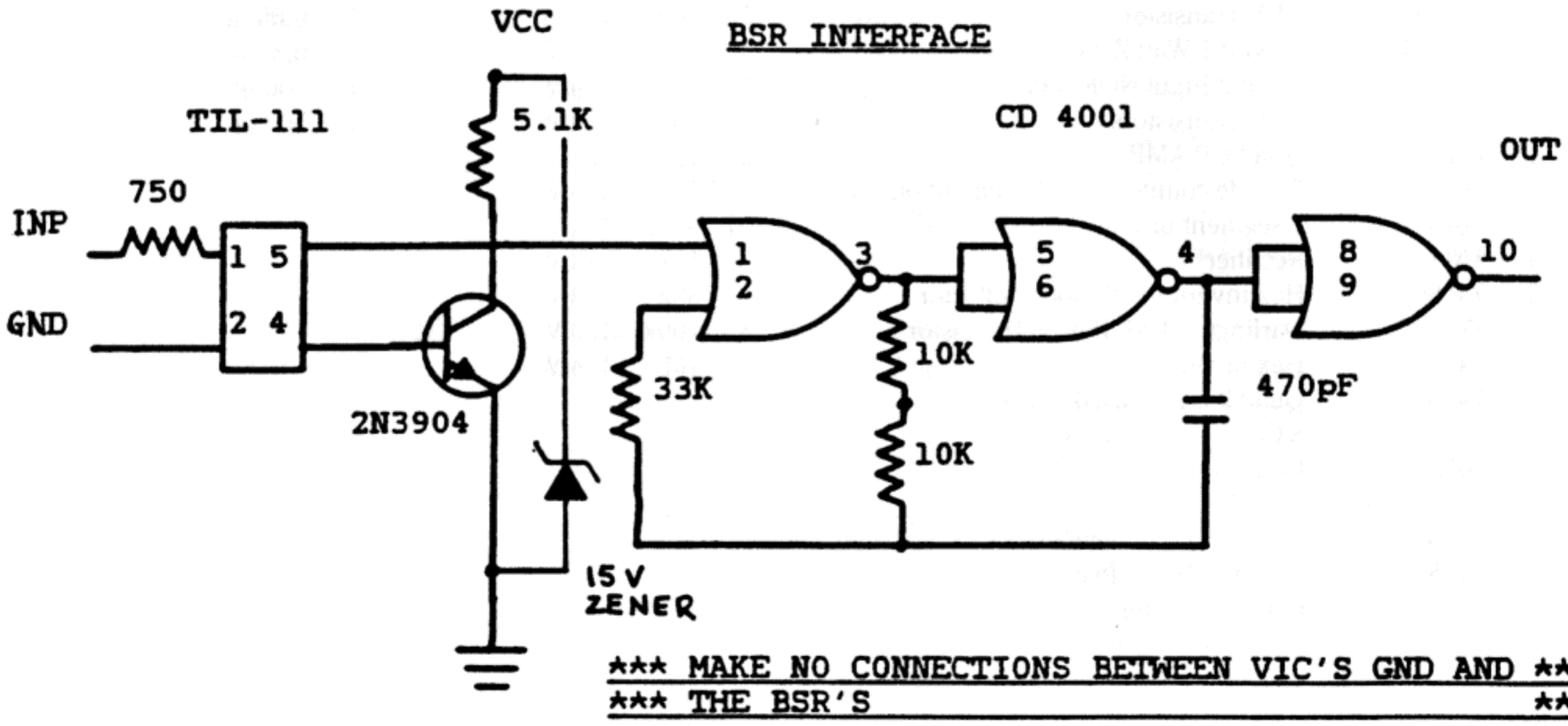
SINGLE CHIP 8K MEMORY EXPANSION



Chip used is a HM6264P-15 by Hitachi

FIGURE 2

BSR INTERFACE



*** MAKE NO CONNECTIONS BETWEEN VIC'S GND AND ***
*** THE BSR'S ***

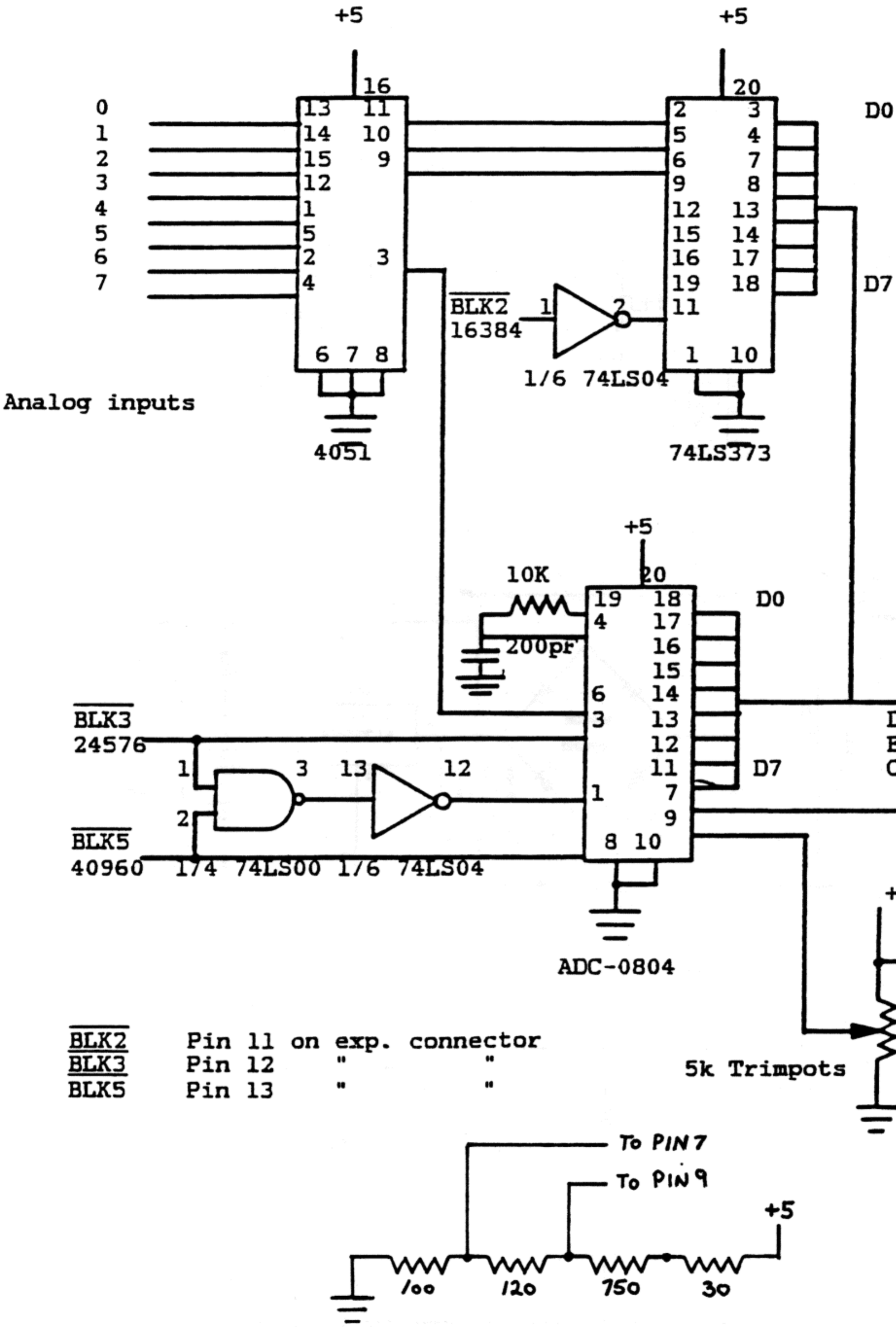
Optoisolator TIL-111
I.C. CD 4001 VCC to pin 14 VSS to pin 7

Oscillator output to BSR pin 7

INP to VIC user port pin C
GND to VIC user port pin A

FIGURE 3

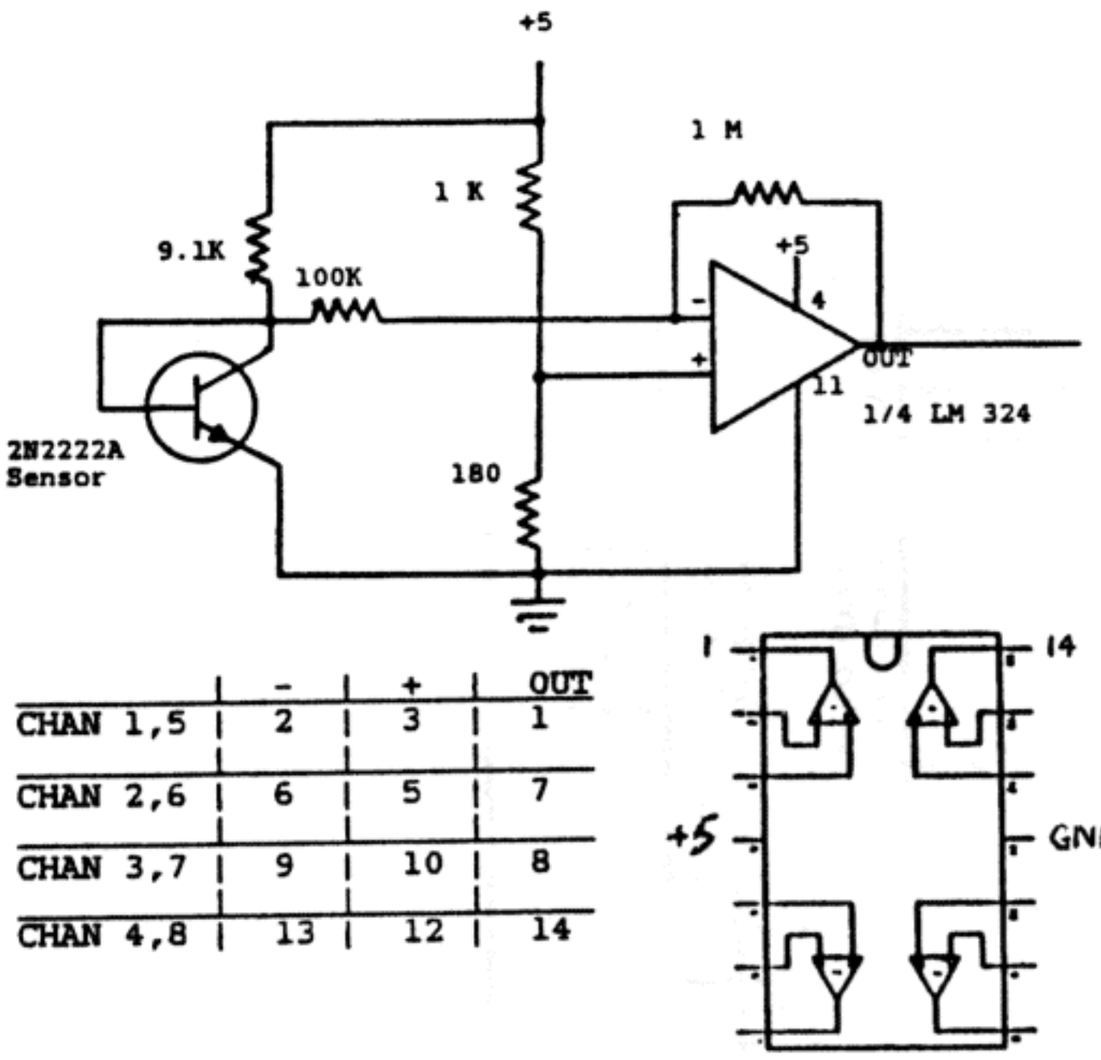
ANALOG TO DIGITAL CONVERTER



BLK2	Pin 11 on exp. connector		
BLK3	Pin 12	"	"
BLK5	Pin 13	"	"

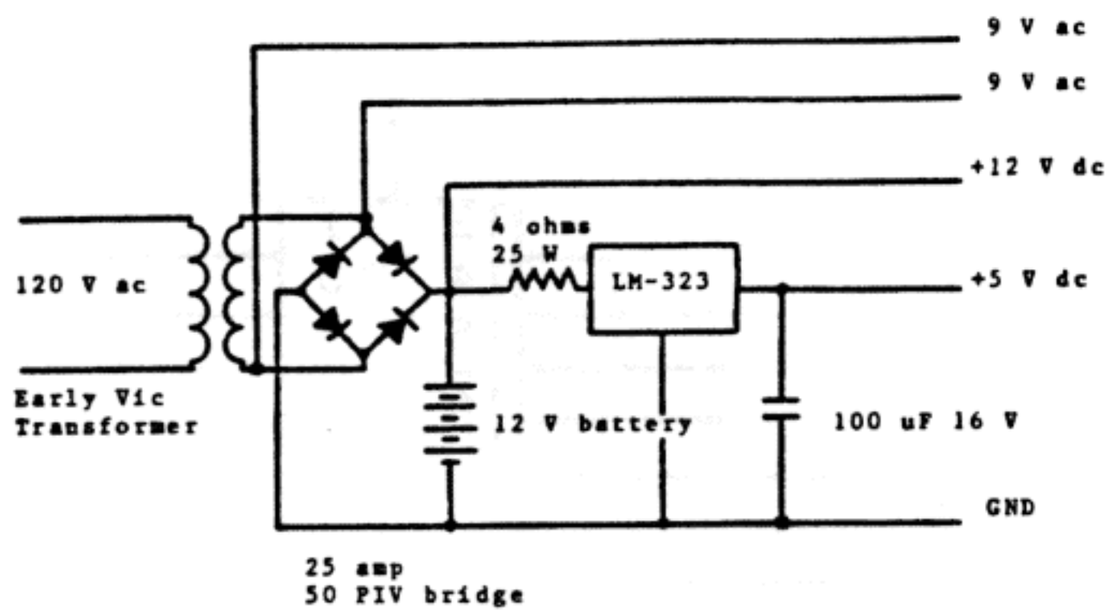
FIGURE 4

TEMPERATURE SENSORS AND AMPLIFIERS



	-	+	OUT
CHAN 1,5	2	3	1
CHAN 2,6	6	5	7
CHAN 3,7	9	10	8
CHAN 4,8	13	12	14

FIGURE 5



The specs for the early VIC transormer are 11 volts A.C. output at 3 amps.

On The new style VIC20's use the GND, 5 volts and the two 9 volts AC leads. On the old style VIC20's use only the GND and 12 volts. The 12 volts is connected at the same point as the alcaline cells as shown in figure 6.

FIGURE 6 ALCALINE CELL BATTERY BACK-UP FOR OLD STYLE VIC'S

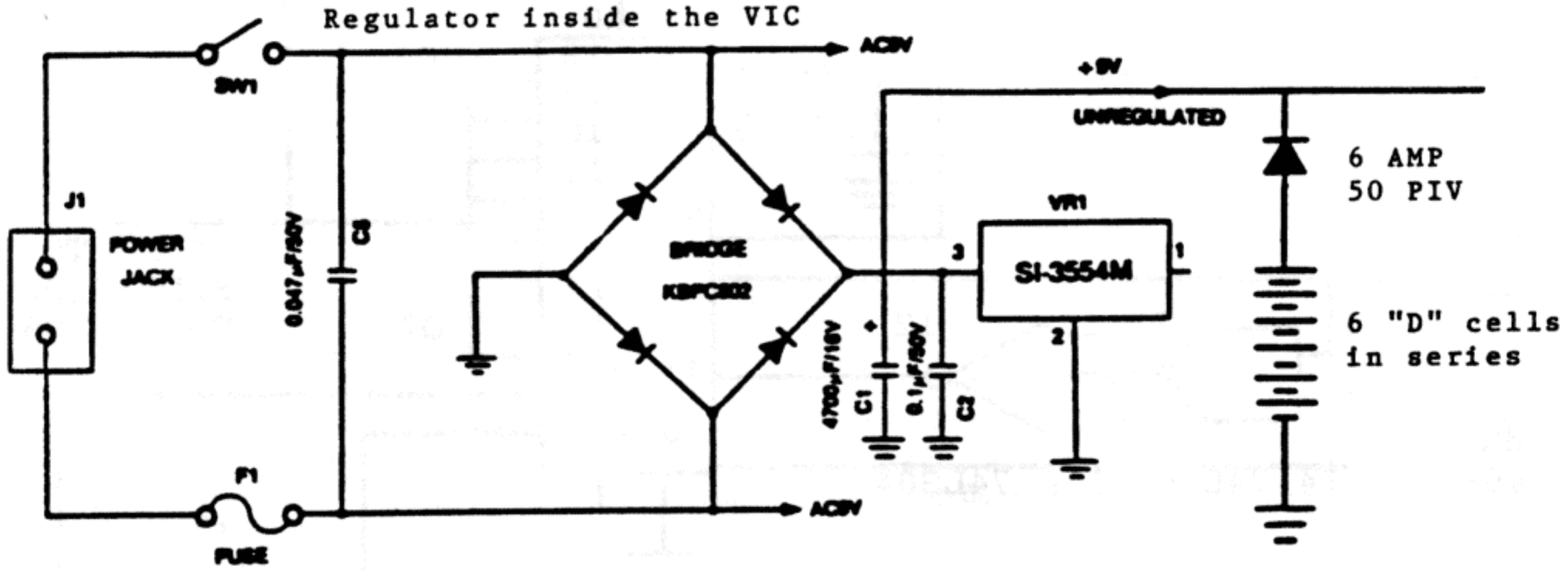


FIGURE 7 ALCALINE CELL BATTERY BACK-UP FOR NEW STYLE VIC'S

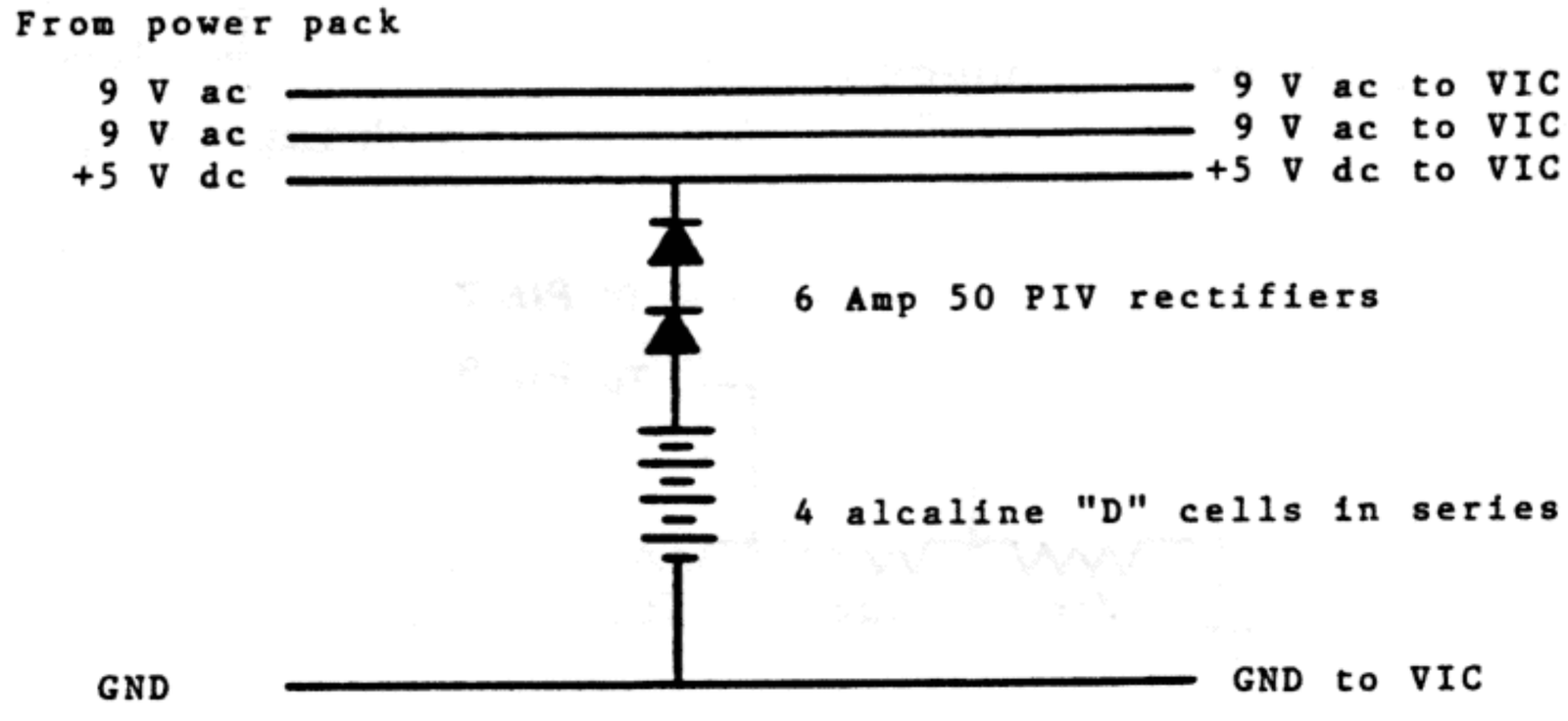
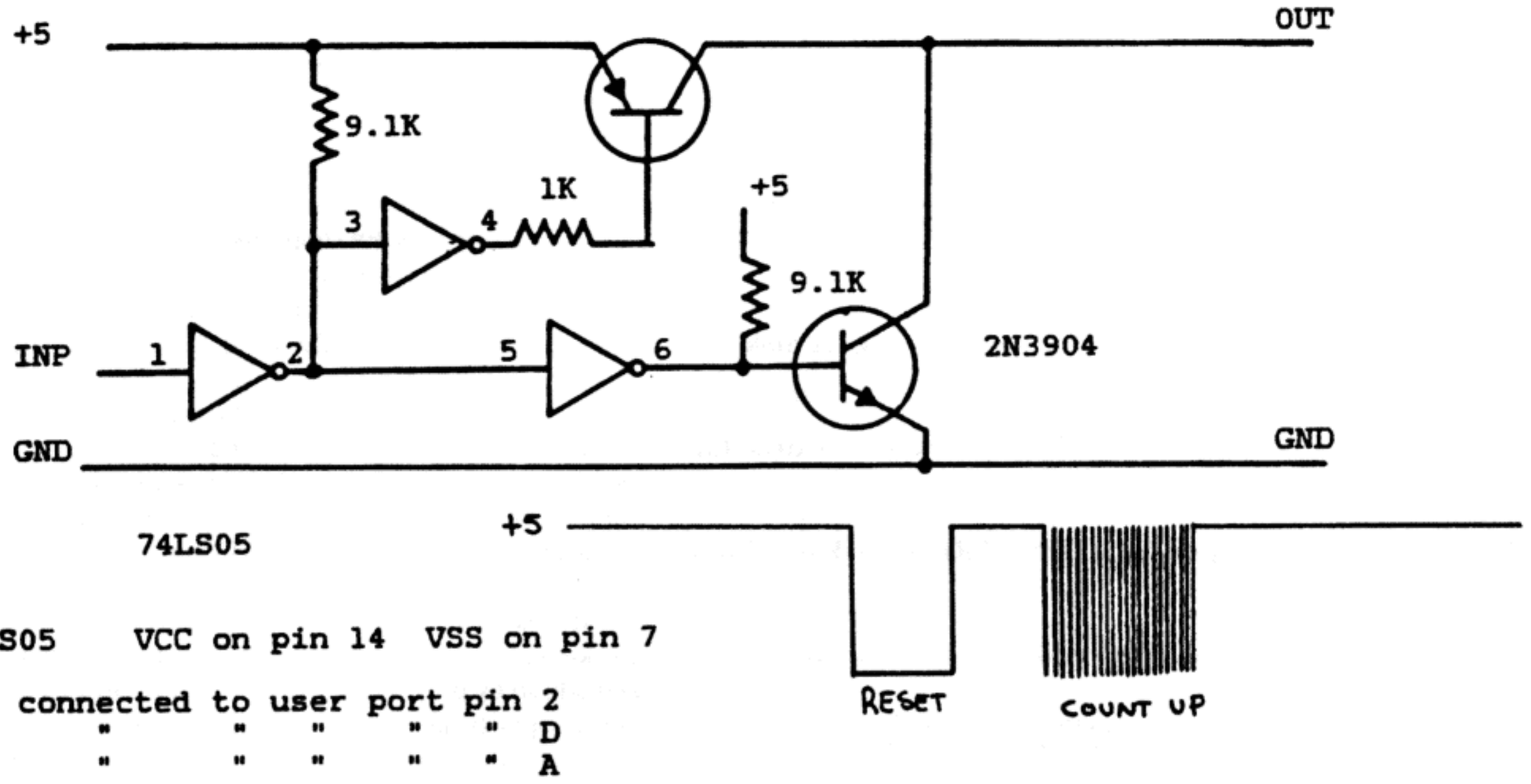


FIGURE 8a

REMOTE DISPLAY TRANSMITTER

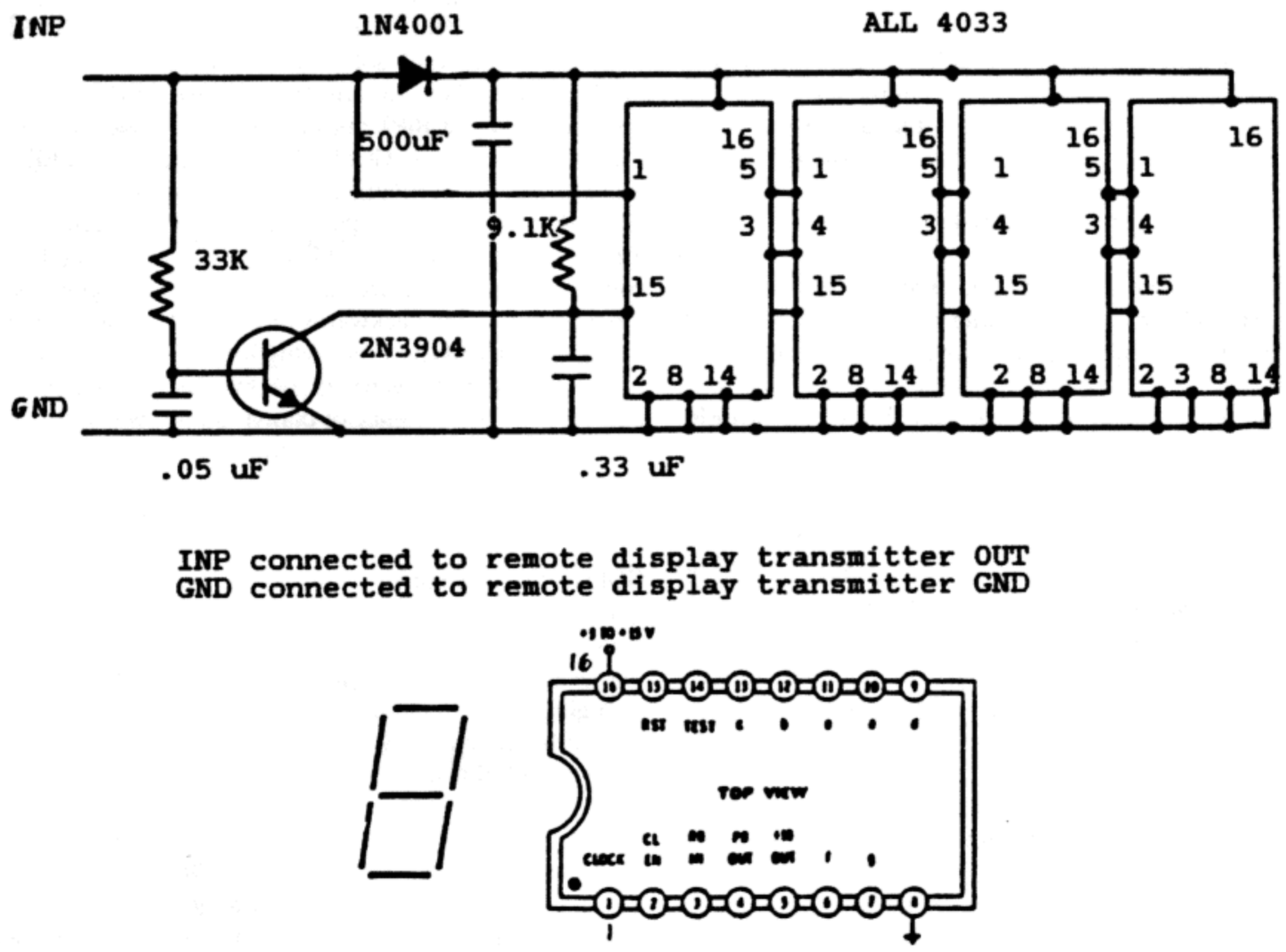


74LS05 VCC on pin 14 VSS on pin 7

+5 connected to user port pin 2
INP " " " " " D
GND " " " " " A

FIGURE 8b

REMOTE DISPLAY RECEIVER



INP connected to remote display transmitter OUT
GND connected to remote display transmitter GND

A Comparison of Four Word Processors

Ranjan Bose
Winnipeg, Manitoba

Even hard-core critics of 'the home computer revolution' grudgingly concede that wordprocessing is reason enough for having a computer around the house. Wordprocessing programs for the Commodore 64 abound in plenty and run in price from a few dollars to hundreds. Presented here is my impression of four medium-priced wordprocessors for beginners and moderately advanced users of wordprocessing. These four programs are:

SPEEDSCRIPT 3.0 from Compute! Publications on disk for $18.00

Gold Disk volume 2 from Gold Disk Software on disk for $16.95

OMNIWRITER/Omnispell from HESWARE on disk for about $50.00

SUPERTEXT-64 from MUSE on disk for about $60.00.

These form two comparable pairs in their prices and functions.

General features: Both SPEEDSCRIPT and GOLDDISK come on write-protected disks with other programs (games, utilities etc.). SPEEDSCRIPT is transferable while GOLDDISK is copy-protected. Instructions for SPEEDSCRIPT appeared in Compute! (vol. 7 no. 3, March 1985), while those for GOLDDISK are included as text files on the same disk. SPEEDSCRIPT and OMNIWRITER both can work with tapes as well as disks. The other two can only access disk.

OMNIWRITER comes with a compact and lucid manual. The system disk is write-protected and contains several example files. The program permits mail merge, label making and has a 30000+ word spelling checker which is adequately fast and is user-editable (add or delete words). There are also utilities for backing up disks (text files) and for copying the dictionary to another disk.

SUPERTEXT comes with two identical disks which can be used for storing files. The manual is roughly the same size as the C64 manual and is excellent. Several example files are provided on the disks.

If you accidentally reset the computer, you can still access SUPERTEXT and SPEEDSCRIPT (document preserved). Not so with OMNIWRITER and GOLDDISK. Both OMNIWRITER and SUPERTEXT have help menus.

Display: Both SPEEDSCRIPT and GOLDDISK have 40 column unformatted displays (not showing margins, paragraphs and page breaks). SPEEDSCRIPT has an on-screen print preview option while GOLDDISK does not. SPEEDSCRIPT can send files to disk as formatted sequential files which can later be directly printed through a GET#/PRINT# loop without using SPEEDSCRIPT. GOLDDISK is the only program in this group of four which does not word-wrap or parse (broken words at right margin). All four allow changing of screen and character colors.

OMNIWRITER uses normal sized characters and a rolling-writer display either in 40 column width or full width (up to 240 columns). The latter shows a portion of a file through a 40 column wide window. The edges can be seen by horizontal scrolling or by pressing F3/F4. A status line at the top of the screen indicates the name of the document and the page, line and column position of the cursor. The display is formatted (except for line spacing and right justification). Print previewing on-screen is not supported. Display is divided into pages and you cannot continuously scroll from one end of a multi-page document to the other. Free movement within a page is possible. F1 moves to the next page; you can also go to any page directly.

SUPERTEXT accepts lines of up to 132 characters and has either a normal sized 40 column display or a hi-res 80 column alphanumeric display; the display is selected by LOADing separate utility modules without affecting the document in memory. The 80 character display is unusable for composing text unless you use a high resolution monochrome monitor, and the manual gives a clear warning about this. It is however, more than adequate for print previewing and for checking page layout. The Display is not normally formatted but can be made so by pressing the '&' key. Print previewing on the screen is highly sophisticated (see printing). There is also a split screen option which allows you to view two individually scrollable parts of the document on the same screen, very helpful during block move and copy operations or while trying out different sentence constructions for easy readability.

Editing and Formatting: SPEEDSCRIPT SAVEs text files as screen codes (PRG). GOLDDISK can SAVE and LOAD ASCII files either as PRG or SEQ. A GOLDDISK file can be LOADed by SPEEDSCRIPT and only formatting and some text characters have to be changed. SPEEDSCRIPT files however look strange on GOLDDISK and cannot be used. SPEEDSCRIPT permits display of disk directories and sending DOS commands. GOLDDISK does neither. SPEEDSCRIPT uses inverse letters as formatting symbols for selecting margins (left, right, top and bottom), page length, width (up to 255 columns), line spacing, forced page breaks, text justification, and centering. It uses keyboard control sequences for transposing letters, changing case, paragraph indenting, deleting backwards or forwards – a letter, word, sentence, or paragraph and then retrieving it if necessary. It also has multiple keys for moving forwards or backwards through the document by character, word, sentence or paragraph. It does not support right and left justification at the same time, though. It supports up to 255 character long headers and footers. It also permits remarks and notes which are displayed on the screen but not printed on paper.

GOLDDISK uses conventional BLOCK commands. You first mark a block and then either move, copy or delete it. Retrieval of deleted material is impossible. It supports all of the above formatting commands except for setting of document width, transposing letters and changing case (nor do the higher priced programs!). The cursor can be moved bi-directionally by a letter, word, line or 16 lines or to the extremes of the document. Both programs have an insert mode (subsequent text moved down while adding text) or replace mode (overwriting). Keys can be defined by both programs (graphics and other special printer codes). SPEEDSCRIPT allows one to print the upper/lower case character-set CBM graphics symbols. None of the others permit that fully. Gold disk does not support headers or footers.

SPEEDSCRIPT has a large buffer to accommodate 44K of text and allows linking of files from disk and tape during printing (not during editing; only SUPERTEXT permits that). GOLDDISK allows only 24K with no linking of files. Both programs allow appending of files in memory by serially LOADing documents. Both allow selective and global searches and replaces. Tabulation is difficult on both if not impossible.

OMNIWRITER files (PRG) are divided into four areas called pages – a work area for notes to yourself, or for moving deleted blocks (sort of an inconvenient recall buffer), and for material which can be merged during printing (addresses, labels etc.). The other three areas are for the text, header and footer. Incidentally, any of these can have more than one page and can be accessed easily for editing. Horizontal tabbing (left justified for text and decimal justified for numbers) makes tabulation easy. Any number of tabs are permitted. The two extreme markers on a format line decide the left and right margin. If these tabs are repositioned later, the entire display changes and text is rearranged. One interesting touch is that when you place the cursor on any format symbol, a normally invisible letter shows up to indicate the key for selecting the function represented. Up to ten special ASCII printer commands can be programmed. Right justification is controlled during printing which means that your entire document will either have a straight or a ragged right margin. Cursor movement to top and bottom of document, by pages, screens, lines, tabs and characters is supported. Other standard features like selective/global searches, replaces, and block operations are provided. Line spacing can be changed from 1 to 9 but does not show up on the screen (same as the other programs). Paging (text length) has to be manually determined by placing end–of–page markers. This also means that if you use variable spacing in your document you have to do a bit of mental calculations since the status line would show line numbers as if it was a single spaced–document and you may end up with a printed page which extends into the next page. You can overwrite or insert text anywhere in the document easily. The lack of automatic paging however has one benefit. It avoids printing a heading belonging to the paragraph starting on the next page at the bottom of a page! The document can be up to 34K long and linking or merging of other files is easy during editing. This includes any sequential file (data bases, spreadsheets, telecommunications) or other SEQ or PRG wordprocessor file. Variable information can be merged from the work page, other OMNIWRITER files, or any sequential file. A form letter thus could be composed and different names and addresses merged with it to get multiple copies (Mail–merge). Directory and DOS commands are a keypress away. If you try to quit without saving an altered document a warning is displayed.

SUPERTEXT is a program with a unique personality. On the one hand it has a very flexible and sophisticated printing package and file linking and merging facilities, yet on the other hand it has a very slow and inefficient editor. It has three operative modes. Pressing F1 toggles the CURSOR mode which is used only for disk access and block operations. The screen goes dead and you cannot type in any text. The directory is displayed in two columns with numbers which can be used in lieu of file names for LOADing. The available space on disk is displayed as pages (roughly corresponding to a double spaced printed page) rather than as the more familiar blocks. The directory displays only USR and SEQ files. While carrying out block operations the marked block can be SAVEd to disk and can be retrieved later if necessary by merging. This method of block retrieval however is cumbersome compared to using the replace–buffer available in SPEEDSCRIPT. SUPERTEXT, like OMNIWRITER, allows non–destructive merges. This means that the material from disk is inserted at the cursor without overwriting the existing document.

Pressing F3 toggles the ADD (insert) mode which alone can be used for composing and editing text. Pressing F5 toggles the CHANGE mode which permits overwriting. You can select the direction of movement of the cursor by pressing + or – and can move by a character, word, half a line, a line, half a page or more. All standard formatting is supported. You can number pages at the top or bottom, in the center of a line or on the right and left edge on alternate pages. Up to 15 tabs are supported for tabulation (right or left justified; for numbers you have to put the 00 after decimal; this is not necessary with the decimal tab of OMNIWRITER). The text area available is only 10K but you can go to a linked file forward or backward and edit it. Also, if you are LOADing a sequential file (composed on some other wordprocessor) which is larger than 10K you can LOAD it in chunks and SAVE them as linked files! Nine user–definable keys are available for sending ASCII codes to printers. You can also define the \ key to represent a frequently used word or phrase up to 30 characters long. Every time the \ key is pressed the phrase is inserted in the document.

Printing: SPEEDSCRIPT allows previewing on screen, GOLDDISK does not. Both print to the lowest common denominator i.e., a dumb printer. SPEEDSCRIPT has problems with RS232 printers. GOLDDISK has problems with a 1526! (a special version is available though).

OMNIWRITER formats the text on–screen and therefore does not need print previewing. You cannot see line spacing or right justification on the screen though. You can print a sheet at a time or use fan fold. Printing can be halted and then you can either continue, reprint the page or abort. Selective printing (e.g. pages 6–19) is possible. Many printers including RS232 devices and the 1520 plotter are supported. There are utilities and special versions for parallel and IEEE printers as well. SUPERTEXT uses customizable separate printer files. Print previewing on screen is supported. Printing on paper and screen can be halted and you can continue, reprint a page, print a line at a time, skip a page or abort. You can print more than one copy (not possible with linked files for obvious reasons). Printing speed is much slower than any other program (it probably uses a different algorithm).

Both these programs can send special codes to trigger printer functions like italics, emphasized, bold, super/subscripts, custom–graphics and underlining etc. (provided your printer supports these functions).

Overview: SPEEDSCRIPT for its price is an extremely sophisticated wordprocessor. It has the largest text area, very flexible and intelligently designed cursor movements (by elements of text rather than by lines/screen etc.). It also has an undo or retrieve function which none of the other packages reviewed here possess. At its price, features like 80 columns, mail merging, or spelling checking should not even be expected. The only vital thing missing is right justification. GOLDDISK has many more shortcomings – a limited text area, no DOS or directory access, no screen previewing capability. It however can right justify. Undoubtedly, SPEEDSCRIPT is an easy winner in this category. It is a serious contender even when compared with the higher priced programs.

The higher priced programs are difficult to compare. SUPERTEXT has a very powerful and flexible printing package (to screen and hard-copy), and extremely powerful text merging and linking/splitting. It however does everything more slowly than the other programs and it has the most inconvenient editor I have ever seen, which is its major drawback. A wordprocessor should leave your mind free to think while you compose a document. SUPERTEXT can slow you down.

The only minor inconvenient feature of OMNIWRITER is its non–automatic page–formatting. When your document is anything but single–spaced, the line # indicator in the top status line is incorrect.

You have to mentally count where on the printed page you are and set page end markers accordingly. Choose variable spacing and you are in for more work! However, it is comforting to know that the feature is there and you can space your document variably up to 9 spaces between lines. SUPERTEXT does not allow this choice. You either have a single or double spaced document throughout, period. This beef aside, OMNIWRITER/Omnispell is a very convenient, efficient package, easily the best among the ones reviewed here. Its flexible formatted screen output, uncluttered screen, alphanumeric tabbing, convenient editing and the icing in the form of mail merging and spelling checking make it a very attractive package. This is a wordprocessor which even a beginner can use easily and continue exploring and growing with it for a very long time.

Additional Note For 1526 Owners: While I was exploring these wordprocessors, I was amazed to find that most software designers had not given serious consideration to the 1526 printer. This is ironic because 1526/MPS–802 is more suitable for serious wordprocessing than the 1525/MPS–801! The 1526 when first introduced had firmware bugs. It would lock up the serial bus and was sensitive to the order in which devices were switched on. A letter from Commodore indicates that the latest C64–compatible 1526 had a version 05 ROM and that the more recent version 07C was introduced to increase compatibility with the newer +4 and +16 computers. My recent experience however indicates that most wordprocessors are allergic to the 1526 with version 05 ROM. You would either get no printout (with GOLDDISK, unless you use a special version, readily supplied by GOLDDISK), strange "?h?HG$–" characters at the head of your document (almost all wordprocessors) or even a word jumble (LETTER WIZARD, DATASOFT). If you have a version 07C ROM all of these will work perfectly. BUT if you RUN any other program which uses multiple custom characters you are in for a surprise! Your programs which worked fine with the 05 ROM will now report a terminator error! The solution is simple. Normally when designing a custom character, you draw an 8 by 8 matrix and add the binary values of columns (1,2,4,8,16,32,64,128) depending on the position of dark printing cells (0 if blank). Thus you have 8 ASCII values which are sent as a concatenated character string to a printer file with a secondary address of 5, and then when you print CHR$(254) you get your custom character. If you have an 07C ROM in your 1526, SEND A 9 CHARACTER STRING TO S.A. 5 USING AN 8 BY 9 MATRIX. The last value does not print and can be a zero. What does this mean? Have the Commodore designers run out of coffee (again?) or psst. . .psst are they going to break the 8 by 8 barrier? Only time can tell. Commodore for sure won't!

Omnispell Spelling Checker: Omniwriter, in addition to being an efficient and affordable wordprocessor, has a spelling checking program with a 30,000 word expandable dictionary both of which can be copied to your work disks. After working on a document one presses the commodore key followed by RUN (SHFT–RUN/STOP). This results in the loading and execution of OMNISPELL. The program first prepares a word–list (about 1–2 min depending on the size of your document and word–distribution). You then have an option of spell–checking, list words alphabetically or by frequency. Spell–checking compares the words with those in the dictionary and marks and displays the unrecognized ones. You then return to your document and issue a verify command. Everytime an unrecognized word is located you have an option to edit it, skip or accept it or to learn the word. Once this is done, you can re–execute OMNISPELL and add any new words learned to the dictionary. This sub–program is reasonably fast and is usually used only once after your document is in final shape. This however makes OMNIWRITER a very attractive choice.

Comparison Table

Advantages

SPEEDSCRIPT	GOLDDISK	OMNIWRITER	SUPERTEXT
Up to 255 columns. Limited CBM graphics. Logical cursor movement. Transpose letter. CHaNge case. Recall buffer. Large capacity. Disk/tape files. VERIFY files. Seq formatted files printable without word processor. Easily copyable. Very good value for money.	Limited CBM graphics. Right justification. Backup utility included. Poor value for money (when considering wordprocessor program alone).	Good design. 240 column display. Good tabulation. Mail merg/label. Disk/tape files. Accepts files from other wordprocessors, any seq. file from data base/spread sheets/tel.com. Parallel/IEEE drivers included. Supports RS232 printers. Excellent warning for altered but UNSAVED files. Backup utility for dictionary and disks. 30000+ words spelling checker. Excellent value for money.	Hires 80 column display. Up to 132 columns display. Split screen edit. Excellent file link. Splitting & link. \ key programmable. Superb print package. Good tabulation. Altered file has a * before filename - no other warning. Includes 2 program disks. Satisfactory value for money (costliest)

Disadvantages

SPEEDSCRIPT	GOLDDISK	OMNIWRITER	SUPERTEXT
40 col display. Destructive merge. No right justification. No altered file warning. Does not support RS232 printers. No backup utility. No mail merge or spell–checking. Difficult tabulation.	40 column display. Destructive merge. No word wrap - only fan fold. No screen preview. No directory/DOS. Smallest effective text area and no linked files. No altered file warning. Disks only. No mail merge or spell–checking. Difficult tabulation. No headers/footers.	Few CBM graphics. Manual paging specially tricky when using variable spacing. Copy protected – head-bumping.	No CBM graphics. Either single or double spacing – variable linespacing not permitted. Inconvenient editing. Only seq files. No tape access. No mail merge or spell-checking. Slow printing. No backup utility.

News BRK

Transactor News

Submitting NEWS BRK Press Releases

If you have a press release which you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way Applicable to Commodore equipment.

The TransBASIC Disk

Well over 100 commands and functions have been published over the last 8 TransBASIC Columns and there are dozens more to come. So we've decided to collect every command, including the unpublished ones, and put them all on the first release of The TransBASIC Disk. A reference manual gives examples of every command in the library. You simply load and run the first program on the disk and begin adding command modules. After assembling the selected modules, your new TransBASIC "dialect" can be saved to disk for future use.

SYMASS 3.0 is the assembler resident on The TransBASIC Disk. It's written in machine code and was modelled after (and tested with) TransBASIC modules. Previously, PAL was necessary for doing the final assembly. SYMASS 3.0 will assemble any source code that is no more exotic than a typical TransBASIC module, but it doesn't output listings or send object code to disk. For development purposes we still recommend PAL (see next item), but SYMASS 3.0 makes the TransBASIC Disk totally self-contained!

The TransBASIC Disk with reference manual is $9.95 (7% pst in Ontario). You can use our postage paid subscription card to order.

PAL and POWER: The ToolBox

Nearly every source code listing in The Transactor is written in PAL format. We often get requests about obtaining the PAL from Pro-Line. But since Pro-Line is a distributor, they would refer requests to a retailer. So to eliminate a little legwork, The Transactor is offering The ToolBox. It contains both the PAL Assembler Development System and POWER, the Basic editor enhancement package. It comes with the disk and two nice manuals. Suggested list price is $129.95. Mail order from us it's just $79.95! And, once again, you can use the order card at center page.

The G-Link Interface

There are a couple of C64 to IEEE interfaces available but one you probably haven't heard much about is the G-Link, or *Glink* as it's pronounced among the few of us who own one. Why do we use Glinks? They're totally transparent! The others have "features" like machine language monitors and Basic extensions that tend to interfere with certain more sophisticated programs. The Glink does none of that. Nor does it use the RAM that lies underneath the BASIC and Kernal ROMs. It also has a switch for serial bus operation. It comes with installation instructions for $49.95, but there are only a few in existence so it's first come first served. Once they're gone, they're gone.

Attention Anthology Owners

The BBS phone numbers section of the Inner Space Anthology lists a phone number for The Simarillion BBS in Garden Grove, CA. The owner of the number informed us that although he has a C64, and would also like to find the correct number for The Simarillion, he is not the number*b* you're looking for. Unfortunately we have misplaced his identity, so, "number*b*,. . . if your're listening,. . . *aren't you ready for the fun and excitement of your own BBS. . ?* Perhaps someone might offer to loan him a modem? and some BBS software? The number is on page 89 of the Anthology, but use the voice line.

Events

World Of Commodore III In Toronto A Success

The third annual World Of Commodore III held in Toronto in early December was a complete success, with more than 33,000 people attending the four-day show. In fact, the show was so successful that Commodore Business Machines is looking at expansion and new features for the next years show.

The WOC is the largest microcomputer show in Canada. This year, Commodore and about 60 other exhibitors displayed, demonstrated and sold Commodore related hardware, software, peripherals and accessories. To enhance the visit, Commodore had C64s, C128s, PC-10s and Amigas available for use and abuse by the patrons of the show. If any questions developed, Commodore employees could always be found lurking about waiting for any opportunity to please. This was surely a noteworthy event in Commodore history!

" Record sales were achieved by repeat exhibitors and those displaying for the first time were amazed at the interest they attracted and the amount of products that sold", said Robert Graham, Vice-President, Marketing. " Virtually all the exhibitors plan to return in 1986."

Along with seminars that often drew more than 350 people, musicians, computer experts, engineers and graphic artists demonstrated the capabilities of the Amiga along with the other Commodore product line. As well, Commodore helped the Ontario Special Olympics for the Mentally Retarded by raising $10,000.00 through the raffle of a C128 system, admissions to a video arcade and the sale of magazines and posters.

C.A.S.E Meeting And Jamboree 1986

The place to be on the weekend of April 26-27, 1986, is at the Opry Land Hotel in Nashville, Tennessee for the C.A.S.E. annual get-together. C.A.S.E. (Commodore Association of the Southeast) is a consortium of the user groups of the southeast United States, formed to better serve the southeastern community of Commodore computer users. We are a non-profit organization recognized by Commodore World, and installed with our own special messaging area on the Commodore National Network, Quantum Link.

We invite the general public to attend our two-day Jamboree. There will be vendors present as well as several guest speakers furnished by Commodore Business Machines, Limited. One admission charge covers the general admission to all areas of the Jamboree.

Tickets are available from any C.A.S.E. affiliate club, or by sending $7.50 (US) to C.A.S.E. at the address below. Everyone is invited to attend this southeastern conference of Commodore computer users. Tickets will be available at the door of the Jamboree 1986 for $10.00.

Commodore oriented user groups may join C.A.S.E. by requesting membership information from C.A.S.E. at the address below. Vendors may inquire as to possible spaces available for the Jamboree 1986 prior to March 1, 1986 through the address listed below.

C.A.S.E., Inc.
PO Box 110386
Nashville, Tennessee 37222

Amiga News

NAAUG For Amiga Users

The North American Amiga Users Group (NAAUG) is distributing its first newsletter as of December 24th. NAAUG has contacted

various users groups and Amiga dealers who have expressed interest in the group. Anyone interested in receiving a free copy of the AmigaHelp Newsletter, containing valuable information and details on how to join, should write:

North American Amiga Users Group
P.O. Box 376
Lemont, PA 16851

NAAUG offers members a wide range of services for the annual membership fee of $25.00 (US), which include; a subscription to "AmigaHelp" newsletter, the Helpline for free one-on-one computer advice, one free disk of public domain software plus full access to the group's Public Domain library, free classified ads to members, bulletin boards, and SIGs. For more information contact:

Richard Shoemaker, Founder
(814) 237-5511, after 4 PM and on wkds.

BBS For Amiga Owners

The system supports up/downloads, message boards, Fidonet mail and more. All interested persons welcome! We have a growing list of public domain software available for download. (Also available by mail from Kinetic Designs on disks for non-modem computerists.)

Call: Casa Mi Amiga, (904) 733-4515, 24 Hours, 16 Meg Online. Or for more information send a SASE to:

Kinetic Designs
Casa Mi Amiga
1187 Dunbar Ct.
Orange Pk, FL 32073

dBx Translator Converts dBASE Programs Into "C".

Desktop A.I. has released a translator that allows moving dBASE programs into the "C" language. The dBx Translator system includes a language translator for processing dBASE source, and a run-time library tool box to replace the dBASE screen handler. Once converted, the dBASE code becomes a fully functional and controllable "C" program giving an application developer complete control over his product and release from the problems and royalty costs of dBASE. In addition, applications can be moved to machines where dBASE is not available such as the AT&T 3B2 under UNIX, Altos under XENIX, and MacIntosh or Amiga systems.

The system is programmed in standard "C" and produces standard "C" from dBASE code. The action of the dBx Translator ranges from complete translation of some dBASE statements to commenting out other statements that have no parallel in "C". The programmer then converts those functions that are too different to allow automatic translation. The run-time library provides the "C" functions that dBASE has built-in and allows dBASE application screens to be functional rapidly without major programming effort.

The system comes with a translation guide book which provides programming tips and translating techniques about changing dBASE code into appropriate "C" code to further assist the programmer. The system is available under MS DOS, (using the ANSI screen handler) UNIX and XENIX, (using the CURSES screen handling packages) and will be available soon for the MacIntosh and AMIGA. The package price ranges from $350.00 to $1,000.00 (US) depending on system configuration. For more information or orders contact:

Desktop A.I.
1720 Post Road East #3
Westport, CT 06880 (203) 255-3400

The Trading Board BBS for the Commodore 64 and C128

The second in the Sure Product line, The Trading Board offers Up and Downloading with New Punter, a time and date clock, auto log on, E-Mail send and read with check at log on, set log off time and lots more. The system package includes a detail manual to help you customize the board to your needs, four programs to get you started, a detail list and a section on advertising your BBS. The system will up and down load with one or two drives and is compatible with most auto-answer modems. For a sample on this amazing system call (805) 492-3668. For more information:

Terry Hill, Creative Enterprises
PO Box 4253, 1714 Sanalwood Pl.
Thousand Oaks, CA
91360 (805) 492-0568

Online News

2890 Databases Available Online

The recently published summer 1985 update issue of the authoritative "Directory of Online Databases" reveals continued growth in the online database industry. With a total of 197 databases going online in the last three months, 2890 databases are now being offered through 442 online services worldwide. This impressive growth rate is slightly offset by a just-less-than record number of databases being dropped by the online services. Accordingly, this loss is credited to the normal housecleaning in the industry in part due to repetitive services offered and a narrow range of subject coverage. According to Dr. Carlos A. Cuadra, president of Cuadra Associates, " ... we don't really perceive this as a major shakeout comparable to that found in other segments of the computer industry. Rather, it is part of the perpetual housecleaning one expects in an industry with a healthy number of entrepreneurs."

The "Directory of Online Databases" is published quarterly and provides accurate and comprehensive coverage of all types of databases that are available to users through online, interactive systems. A one-year subscription includes two complete editions and two update supplements. For further information, contact:

Carlos A. Cuadra, President
Cuadra Associates, Inc.
2001 Wilshire Blvd., Suite 305
Santa Monica. CA
90403 (213) 829-9972

George Novotny, Vice-President
Applied and Information Sciences
Elsevier Sciences Publishing Co., Inc.
52 Vanderbilt Avenue
New York, NY
10017 (212) 370-5520, ext 1537

Tymnet Offers First-Time Local Access Service In Canada

Tymnet, McDonnell Douglas Network Systems Company, has announced it now offers a local async dial-up access to its TYMNET public data network in Toronto, Canada. No other U.S. public data network offers local dial-up services anywhere in Canada.

Now, with a local phone call, Toronto users can access the various data bases available through the TYMNET network, and can take advantage of its numerous value-added features and services including built-in protocol conversion, error-correction, comprehensive network management, and more.

Previously, Toronto users had access to TYMNET only by means of X.75 gateways to Datapac, Canada's largest public data network. (These gateways will remain in place indefinitely, however.)

TYMNET plans to offer local dial-up access in five major Canadian cities by first quarter 1986, and in every major Canadian city by year-end 1986.

" With local access availability, we can now offer Toronto businesses an extremely cost-effective, practical data communications solution for a wide variety of business applications," said Neil Sullivan, Tymnet's Director of External Services. " For example, with a local phone call, Toronto users will be able to access IBM hosts running bisync or SDLC from inexpensive asynchronous terminals."

TYMNET provides local access from more than 540 locations in the U.S. and from 65 countries.

Tymnet Inc.
2710 Orchard Pkwy.
San Jose, CA 95134
(408) 942-5076 Dave Tivol
(408) 942-5209 Lori Waggener

Software

Basic Compiler For The C128

Abacus Software of Grand Rapids, Michigan, has announced the release of a new Basic compiler for the Commodore C-128, "BASIC-128 Compiler". Written in West Germany by Thomas Helbig, the "BASIC-128 Compiler" will increase the speed of Basic program execution by a factor of 5 to 35. For the ultimate in versatility, you can compile Basic programs to either pure 8510 machine code, very compact P-code, or a combination of both. The choice is yours.

Included with the "BASIC-128 Compiler" is an 80-page instruction guide which details all aspects of working with the compiler. Simple to very advanced features are outlined, including tips and techniques that every programmer will appreciate. The suggested retail price is $59.85 (US). For more information, contact:

Abacus Software
P.O. Box 7211
2201 Kalamazoo S.E.
Grand Rapids, MI
49510 (616) 241-5510

Chartpak 128 for the Commodore C128

There's a new Chartpak available for the Commodore C-128, "Chartpak-128". Taking advantage of the new, expanded features of the C-128, "Chartpak-128" now has 3X the resolution of the earlier Commodore 64 version. You can now view an entire chart or graph, or scroll the screen to see the higher-resolution detail. And, utilizing the C128's extra RAM, Chartpak allows you to enter a greater amount of data in which to build your charts, while still retaining those familiar data entry/maintenance features that made the C-64 version so successful. Through Chartpak you can produce high-quality pie, bar, or line charts and graphs that, once completed, can be printed out to a variety of dot-matrix printers.

Included with Chartpak is a 140-page user's guide which contains several tutorials to walk you through the building of charts and graphs. "Chartpak-128" has a suggested retail price of $39.95 (US). For more information, contact Abacus Software (above)

Statistical Programs for Commodores

David J. Pittenger and Milton H. Hodge of The University of Georgia announce the release of statistical analyses programs for the PET 4032, CBM 8032, and Commodore 64. The package contains a utility program, Data Manager, which allows the user to enter, edit and save to disk a file of data to be analysed. All of the computational programs are designed to operate on these data files. The computational programs include:

1) Descriptive Statistics, which calculate the arithmetic mean, geometric mean, harmonic mean, quartiles, quartile range, mode, median, variance, standard deviation, coefficient of variation, and skewness and kurtosis index for each group.
2) Correlation/Regression, which calculates and reports all possible correlations in a set of data, and allows the user to perform multiple linear regression with any combination of variables.
3) Analysis of Variance, which performs an analysis of variance on up to nine independent variables arranged with or without repeated measures. In addition, the program accepts unequal cell sizes.
4) Significance Tester, which calculates the exact probability of 'z' scores, 't' and 'F' ratios, correlation coefficients (r), and chi squared scores.
5) Data Transformer, which allows the data to be manipulated in order to maximize the homogeneity of variance.
6) Random Number Generator, which generates up to 6,000 random numbers within a range selected by the user.
7) Permutations and Combinations, which calculates the permutations and combinations possible using the different adding rules.

The program requires a Commodore computer, disk drive, and printer. Several of the programs have been compiled to optimize computation speed, but all of the original Basic code is also supplied for users who wish to review the algorithm used. A copy of the program disk and an operation manual may be purchased by sending $10.00 (US) to:

David J. Pittenger
Department of Psychology
The University of Georgia
Athens, GA 30602

Bookkeeper's Aid: More Than Just A Tax Record System

Northland Accounting has released the new revised editions of the "Taxaid" series of income tax preparation programs for the Commodore 64, Vic 20 and Plus/4 computers. The programs were written by experienced tax accountants and are designed for home use. The new revised editions include all the latest changes in the tax laws.

TAXAID is easy to use with a detailed manual that leads the user step by step through the data entry. The program is menu driven with advanced editing features that allow the user to make changes and revisions at any time during the data entry process. Data files can be saved and reloaded at any stage of the program, calculations are automatic and all tax tables, including income averaging, are built in. TAXAID will prepare any IRS form 1040. The results can be directed to the monitor or printer. Low cost updates for future years are published yearly.

TAXAID is available on disk or tape for the Vic 20 with 16k, the Commodore 64 and the Plus/4 at a cost of $39.95 (US). For more information contact:

Taxaid Software, Inc.
606 Second Avenue SE
Two Harbors, MN 55616 (218) 834-3600

Hardware

Wilanta Descender ROM

Get true descenders for the Commodore 801, 1525, 803, GP-100, Hush 80 and similar printers with the Wilanta Descender ROMs. Features include a uniform character formation, no change in existing graphics capabilities, no change in software compatibility, and no change in printer operation. With complete instructions for an easy installation with no soldering, only $29.95 (US), $39.95 (Cdn). Ontario residents please add 7% provincial sales tax. Cheque, money order, Visa or MasterCard accepted.

Wilanta Arts
6943 Barrisdale Drive
Mississauga, Ontario
L5N-2H5 (416) 858-9298

Attention B Machine Owners: One Megabyte Of RAM Available

How would all of you Commodore B128/B256 owners like to have a megabyte of RAM in your B machines?

Question -- What can you do with a megabyte of RAM in your B?

Answer -- SUPERSCRIPT II allows the use of banks 2 through 9. That's 8 documents in your computer at one time!

Answer -- The 8432 Emulator program, which allows the B128 to emulate the Commodore 8032 computer, will let you load Basic programs into banks 0 through 14 with a maximum of 12, and allows you to switch back and forth between them. That's 12 programs in your computer at the same time!

Question -- What's involved in the 1 megabyte expansion?

Answer -- The 1 megabyte consists of 32-256k dynamic RAM chips in sockets on the original board in place of the 64k RAM chips that are there now and a custom memory management circuit.

To find out more about the 1 megabyte expansion, contact:

Fred M. King
1804 Plover Spring Drive
Dept. MEMEX B
Plover, Wisconsin
54467 (715) 341-1149

CAPTAIN SYNTAX

©1985 DON SLOAN

SNARL

NO! WAIT! LET'S TALK!

A BRIBE, PERHAPS?

TAKE THIS!

POW

AND NOW FOR FLOTSKY'S FLUNKY!

ONE SECOND, SIR!

OOPS!

S-STAY AWAY! YOU WON'T GET ME! I'LL KILL YOU! I WILL KILL YOU!

Y'KNOW, I DON'T THINK SO!

WOK!

AND NOW FOR FLOTSKY'S COMPUTER! ONCE THIS IS TRASHED, THE MICROCHIPS WILL BE USELESS!

...THE GOOD CAPTAIN DEPARTS.

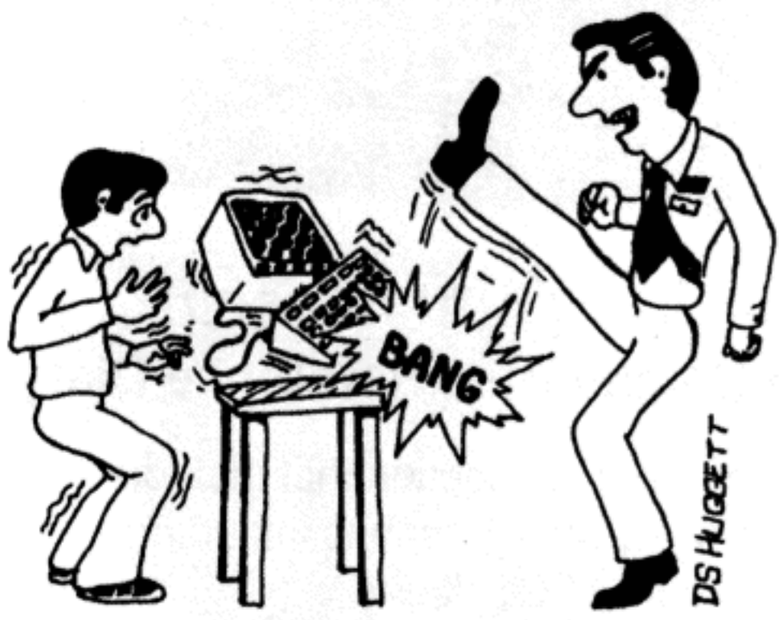
AFTER THE POLICE ARE PHONED FOR A PICK UP...

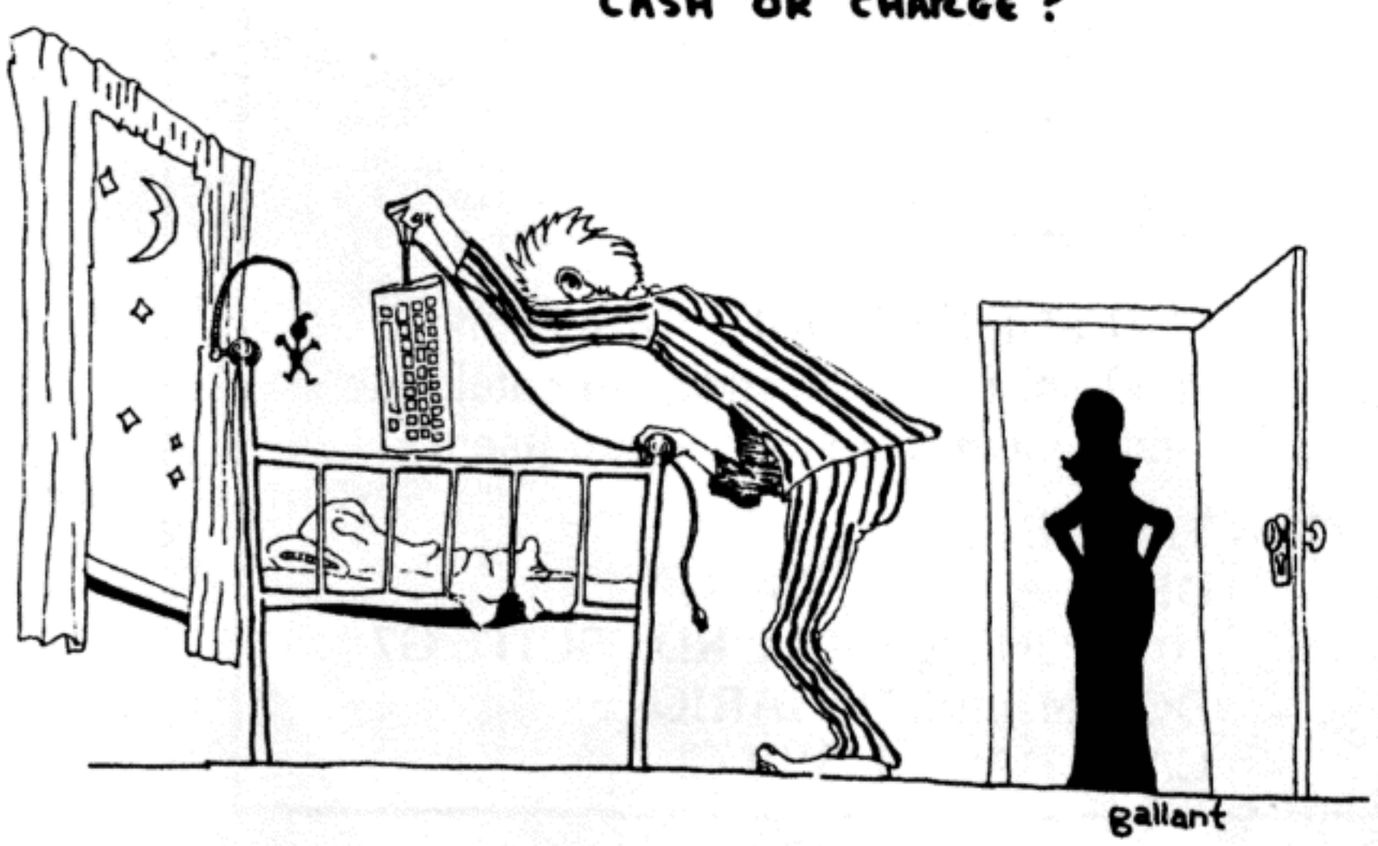
THE END!

Compu–toons

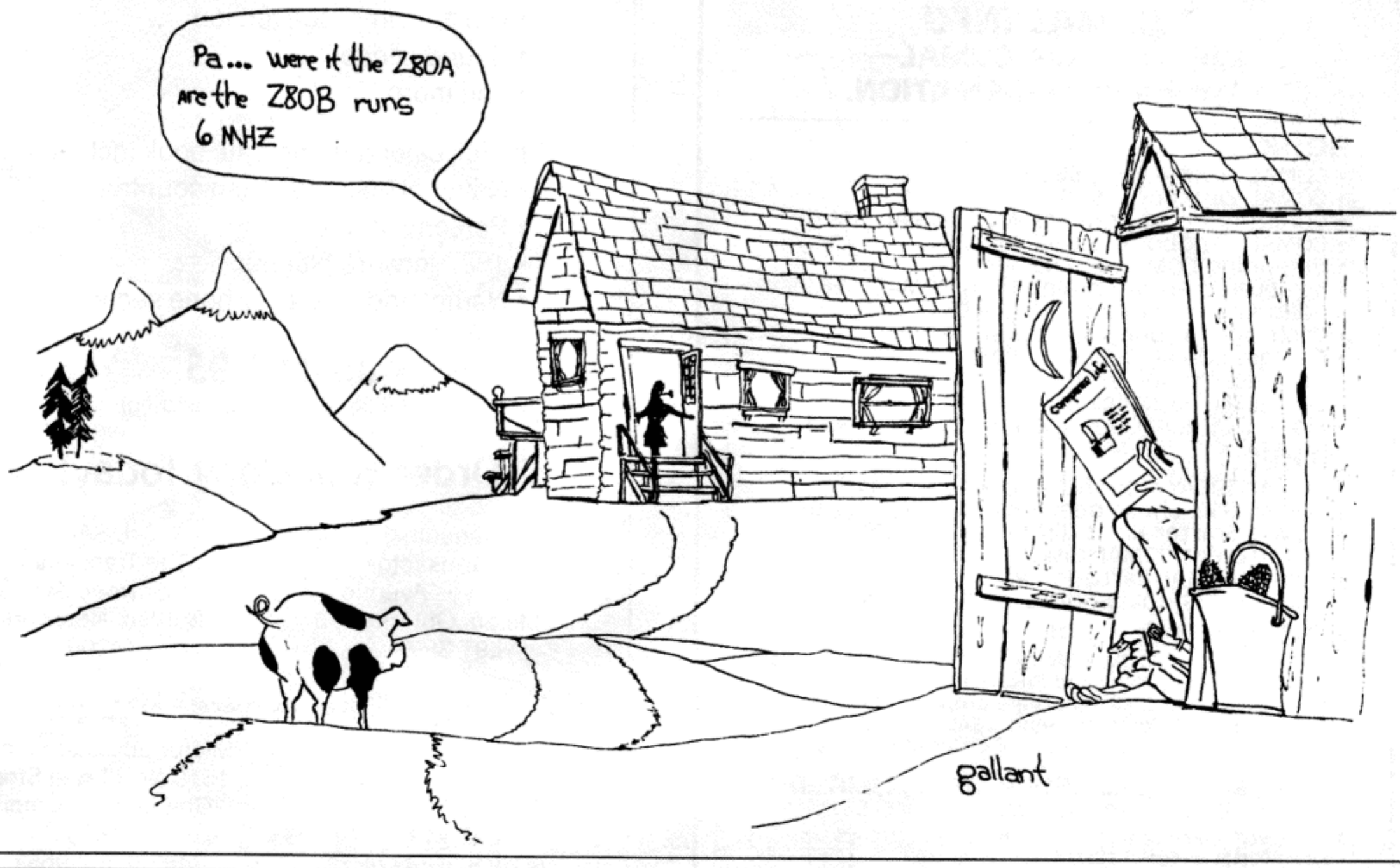
The Transactor has been collecting cartoons for almost two years. Several of them, for one reason or another, will never be published. We're considering the idea of a "Complete Transactor Cartoon Collection" – every cartoon ever received, the best, and the worst, bound together into a low cost booklet that we could also include free with other items or give away at computer shows. If you have submitted cartoons that we have deemed unsuitable for the magazine, we would still like to include them in the "collection". However, we would not be able to pay you for them. Should this be unacceptable, please contact us and we'll have your drawings returned. Otherwise, we'll send a free copy of The CTCC to everyone who participates.

CASH OR CHARGE?





This one's stubborn – it has to be kick-started!



JOIN TPUG

The largest Commodore Users Group

Benefit from:

Access to library of public domain software
for C-64, VIC 20 and PET/CBM

Magazine (10 per year) with advice from

Jim Butterfield
Brad Bjomdahl
Liz Deal

TPUG yearly memberships:

Regular member (attends meetings)	—$35.00 Cdn.
Student member (full-time, attends meetings)	—$25.00 Cdn.
Associate (Canada)	—$25.00 Cdn.
Associate (U.S.A.)	—$25.00 U.S.
	—$30.00 Cdn.
Associate (Overseas — sea mail)	—$35.00 U.S.
Associate (Overseas — air mail)	—$45.00 U.S.

FOR FURTHER INFORMATION:
Send $1.00 for an information catalogue
(tell us which machine you use!)

To: TPUG INC.
DEPT. A,
101 DUNCAN MILL RD., SUITE G7
DON MILLS, ONTARIO
CANADA M3B 1Z3

COMAL INFO
If you have COMAL— We have INFORMATION.

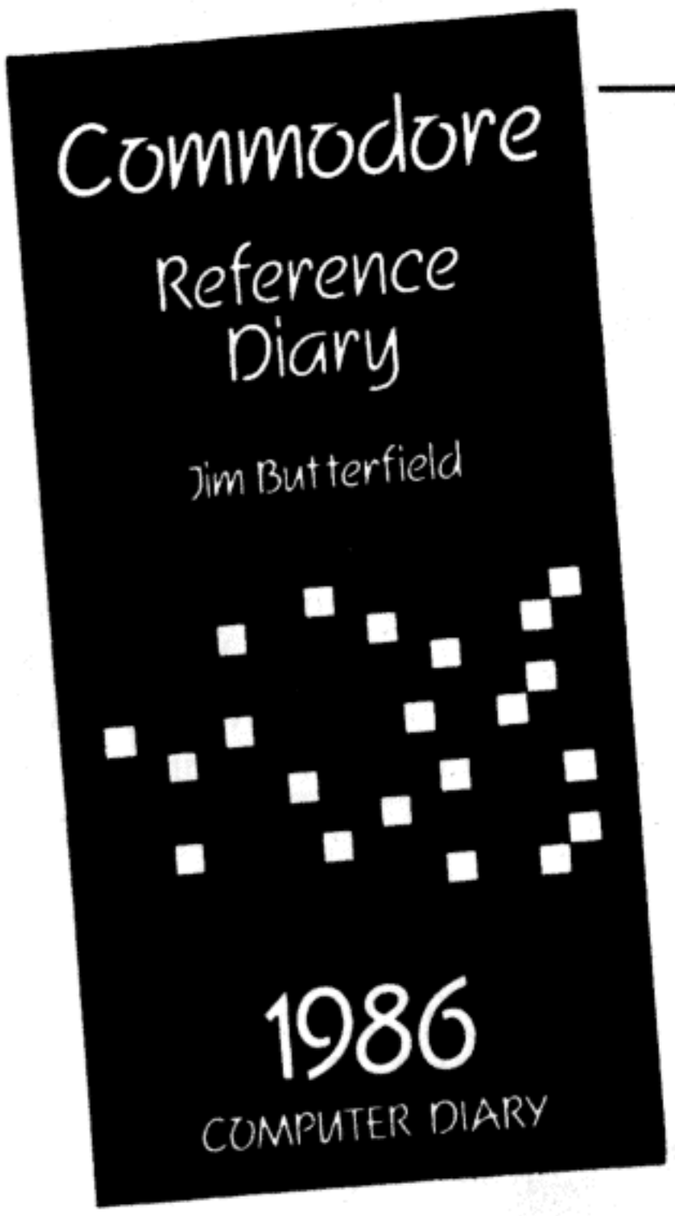
BOOKS:
- COMAL From A To Z, $6.95
- COMAL Workbook, $6.95
- Commodore 64 Graphics With COMAL, $14.95
- COMAL Handbook, $18.95
- Beginning COMAL, $22.95
- Structured Programming With COMAL, $26.95
- Foundations With COMAL, $19.95
- Cartridge Graphics and Sound, $9.95
- Captain COMAL Gets Organized, $19.95
- Graphics Primer, $19.95
- COMAL 2.0 Packages, $19.95
- Library of Functions and Procedures, $19.95

OTHER:
- COMAL TODAY subscription, 6 issues, $14.95
- COMAL 0.14, Cheatsheet Keyboard Overlay, $3.95
- COMAL Starter Kit (3 disks, 1 book), $29.95
- 19 Different COMAL Disks only $94.05
- Deluxe COMAL Cartridge Package, $128.95 (includes 2 books, 2 disks, and cartridge)

ORDER NOW:
Call TOLL-FREE: 1-800-356-5324 ext 1307 VISA or MasterCard ORDERS ONLY. Questions and Information must call our Info Line: 608-222-4432. All orders prepaid only—no C.O.D. Add $2 per book shipping. Send a SASE for FREE Info Package or send check or money order in US Dollars to:

COMAL USERS GROUP, U.S.A., LIMITED
5501 Groveland Ter., Madison, WI 53716

TRADEMARKS: Commodore 64 of Commodore Electronics Ltd.; Captain COMAL of COMAL Users Group, U.S.A., Ltd.



From The Guru Himself!
The 1986 Commodore Reference Diary

A 65 page reference section that includes:
- All hardware specifications including the C128 and PC10/20
- Useful memory locations
- Useful programs
- SuperCharts
- BASIC and machine language hints
- Hexadecimal conversion
- Sound, video
- and more

The full calendar and date book includes:
- National holidays in ten countries
- Personal notes
- 1987 forward planner
- Name, address, telephone section

Just $5.95
(plus 50¢ postage and handling)

Order Your Copy Today!

Canada
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

USA
The Transactor
277 Linwood Avenue
Buffalo, New York
14209

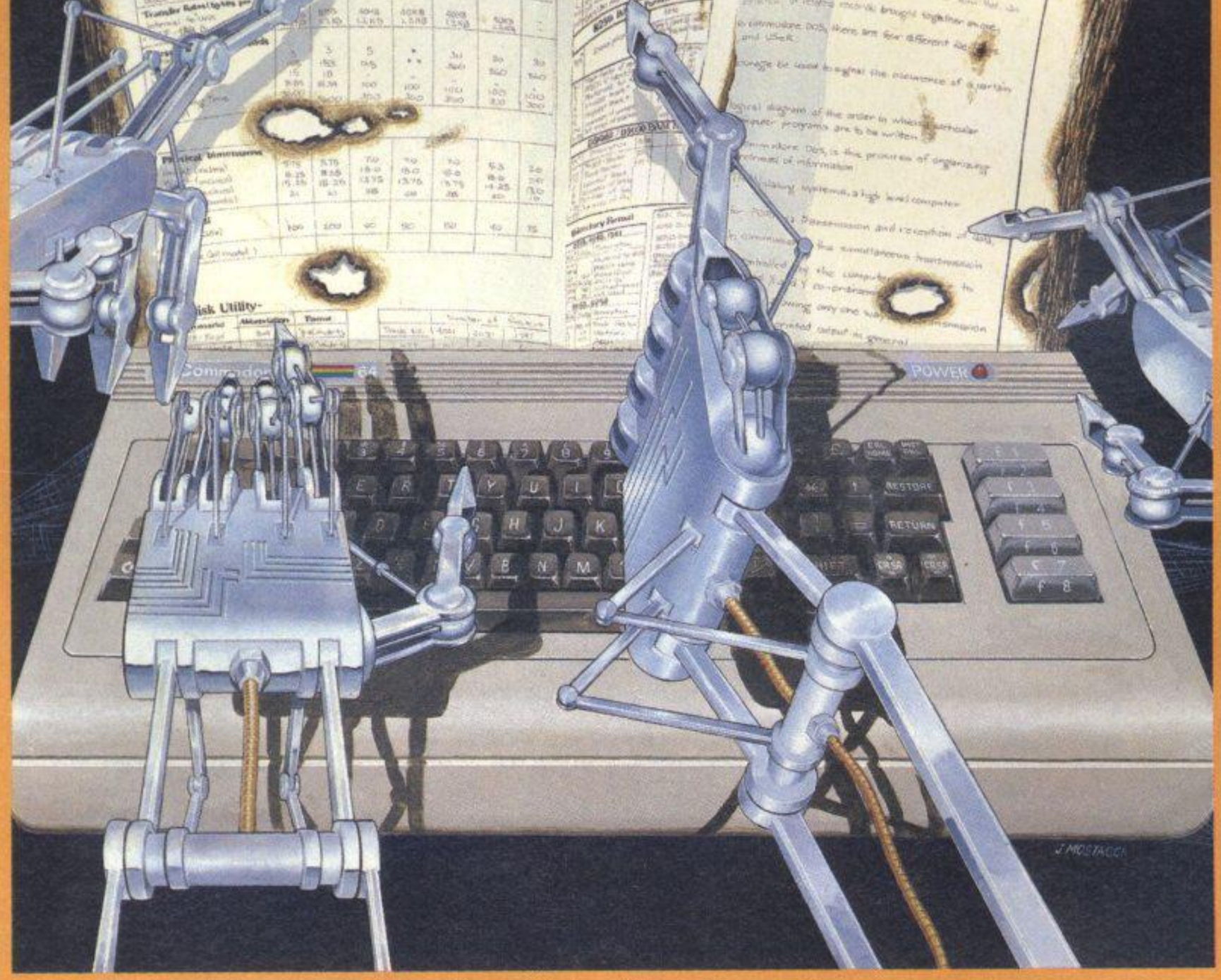
Dealer Orders:

Canada
Norland Agencies
251 Nipissing Road
Milton, Ontario
L9T 4T7
(416) 876 - 4774

USA
MicroPace Distributing
1510 North Neil Street
Champaign, Illinois
61820
1 800 362-9653

Transactor

The Transactor presents, The Complete Commodore Inner Space Anthology



Over 7,000 Delivered Since March '85

Postage Paid Order Form at Center Page